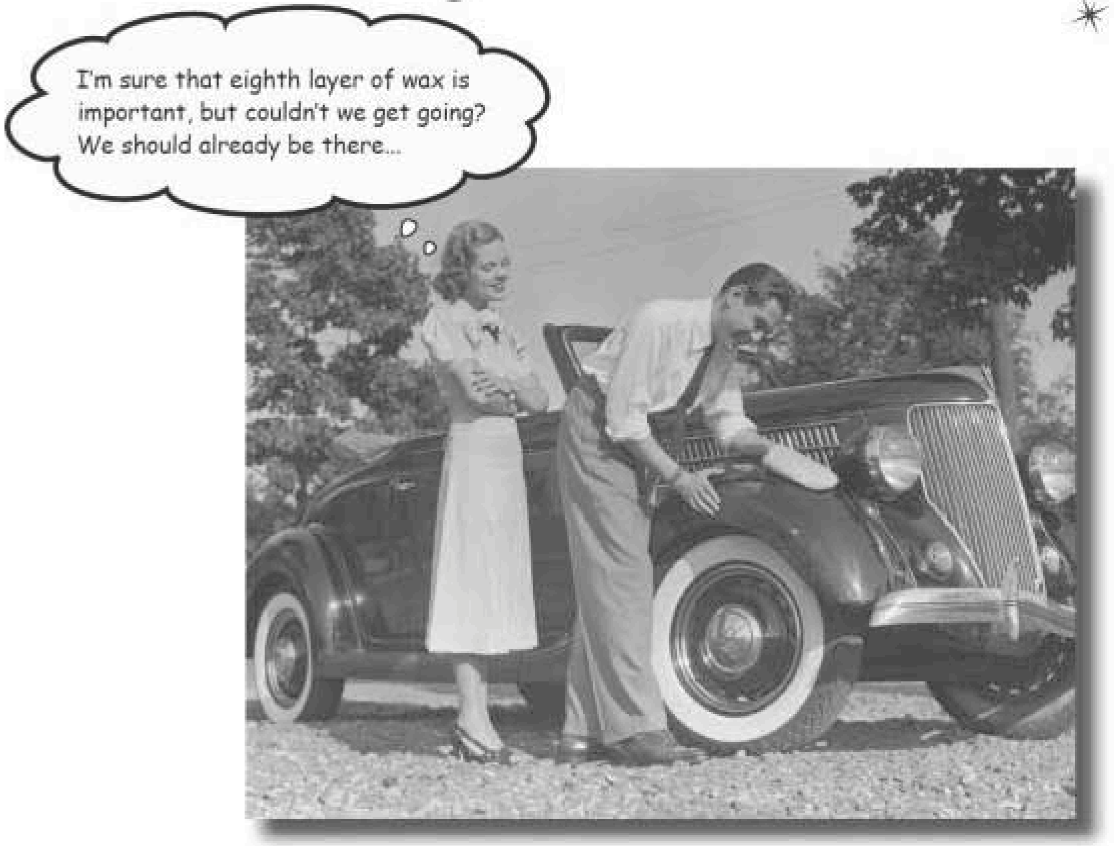
Getting to the real work

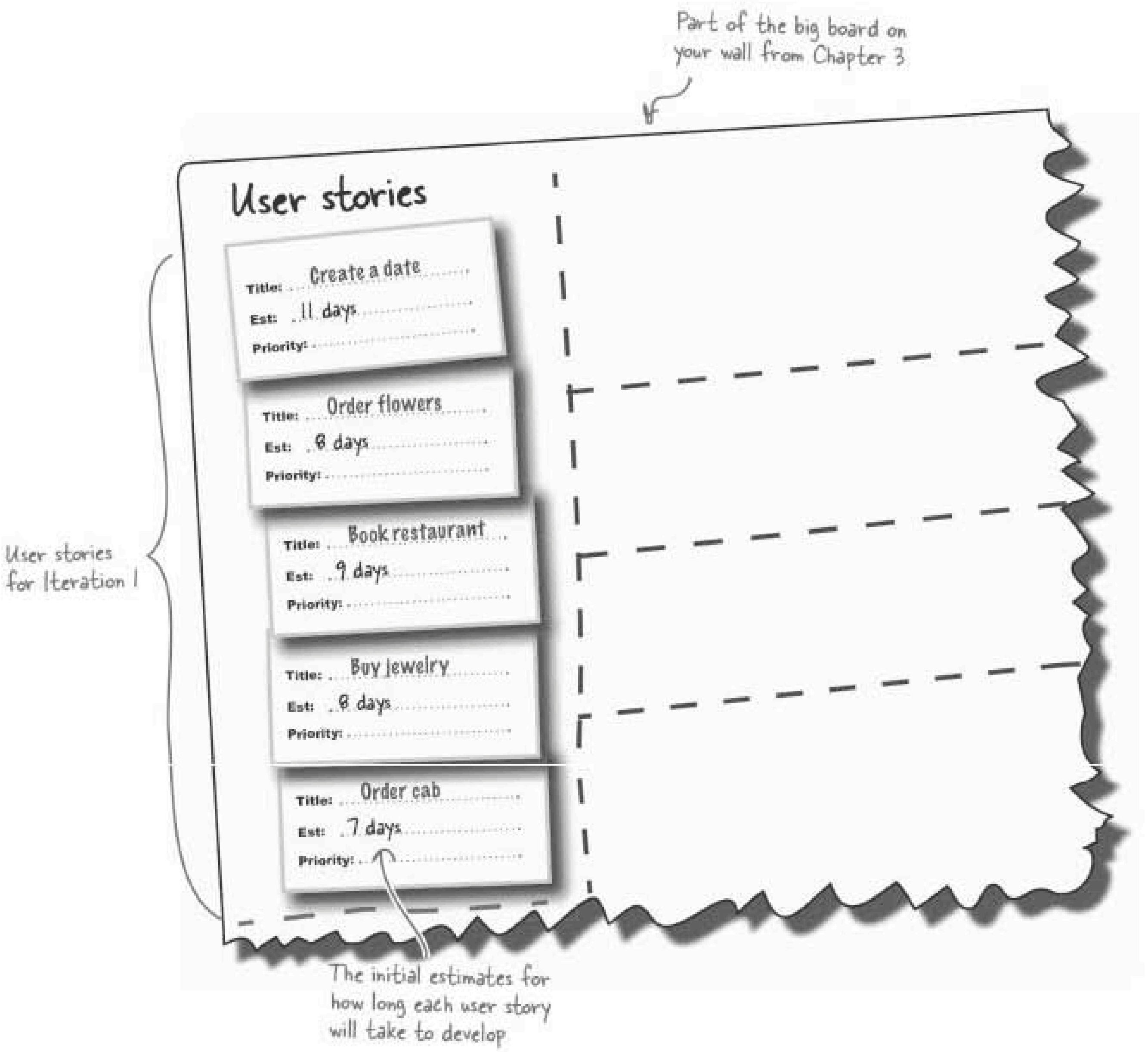

I'm sure that eighth layer of wax is important, but couldn't we get going? We should already be there...

It's time to go to work. User stories capture what you need to develop, but now it's time to knuckle down and **dish out the work that needs to be done** so that you can bring those user stories to life. In this chapter you'll learn how to **break your user stories into tasks,** and how **task estimates** help you track your project from inception to completion. You'll learn how to update your board, moving tasks from in progress to complete, to finally **completing an entire user story**. Along the way, you'll handle and prioritize the inevitable **unexpected work** your customer will add to your plate.

Introducing iSwoon

Welcome to iSwoon, soon to be the world's finest desktop date planner! Here's the big board, already loaded with user stories broken down into 20-work-day iterations:

Date Rights Management (DRM) included



Exercise

It's time to get you and your team of developers working. Take each of the iSwoon user stories for Iteration 1 and assign each to a developer by drawing a line from the user story to the developer of your choice...

Title: Order flowers
Est: 8 days
Priority:

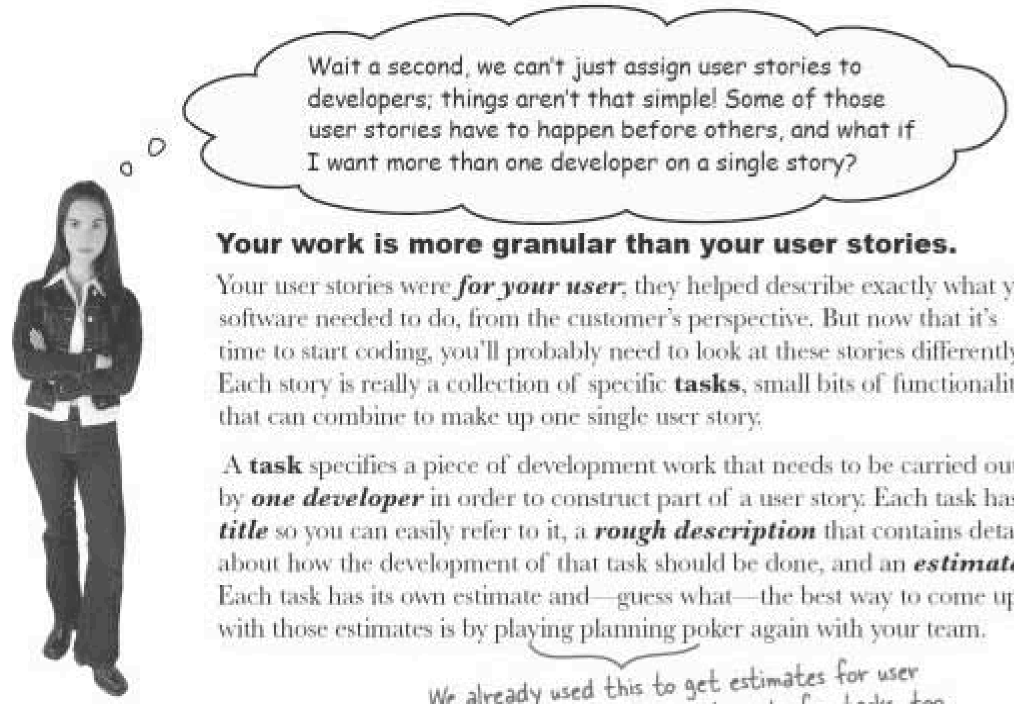
Title: Book restaurant
Est: 9 days
Priority:

Title: Create a date
Est: 11 days
Priority:

Title: Buy Jewelry
Est: 8 days
Priority:

Title: Order cab
Est: 7 days
Priority:

Mark, database expert and SQL black belt
Laura, the UI guru
Bob, the junior developer


Wait a second, we can't just assign user stories to developers; things aren't that simple! Some of those user stories have to happen before others, and what if I want more than one developer on a single story?

Your work is more granular than your user stories.

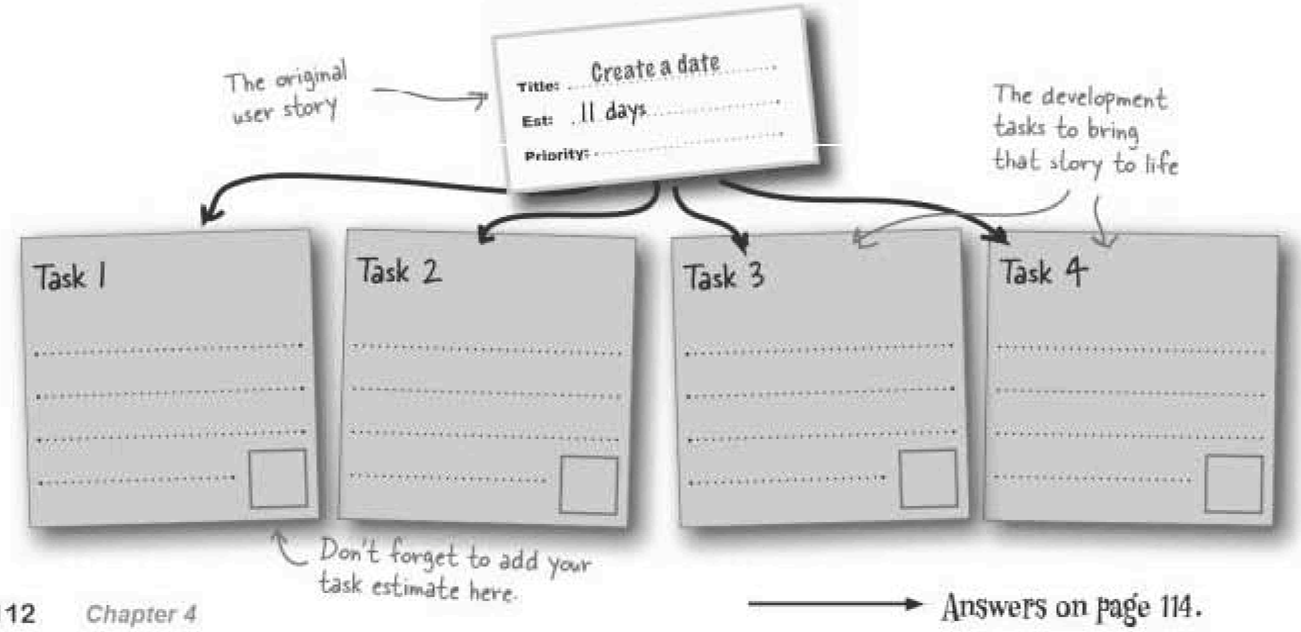
Your user stories were ***for your user***; they helped describe exactly what you software needed to do, from the customer's perspective. But now that it's time to start coding, you'll probably need to look at these stories differently. Each story is really a collection of specific **tasks**, small bits of functionality that can combine to make up one single user story.

A **task** specifies a piece of development work that needs to be carried out by ***one developer*** in order to construct part of a user story. Each task has a ***title*** so you can easily refer to it, a ***rough description*** that contains details about how the development of that task should be done, and an ***estimate***. Each task has its own estimate and—guess what—the best way to come up with those estimates is by playing planning poker again with your team.

We already used this to get estimates for user stories in Chapter 2, and it works for tasks, too.

Sharpen your pencil

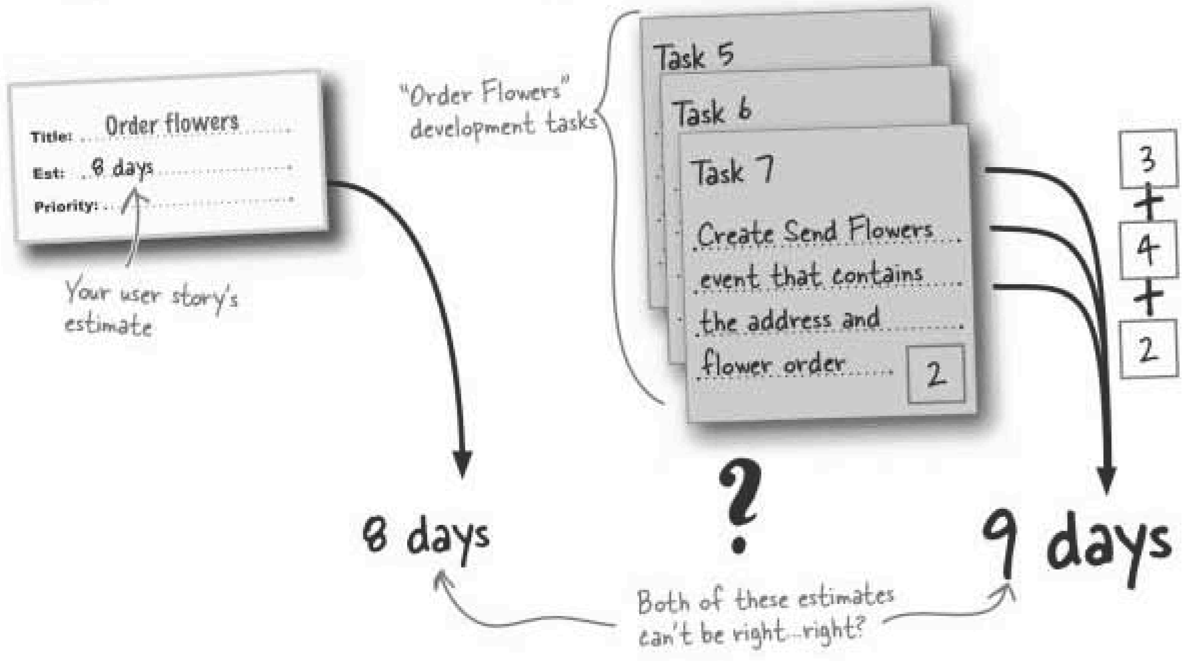
Now it's your turn. Take the user story of creating a date and break it into tasks you think you and your team need to execute. Write one task down on each of the sticky notes, and don't forget to add an estimate to each task.


The original user story
Title: Create a date
Est: 11 days
Priority:
The development tasks to bring that story to life
Task 1
Task 2
Task 3
Task 4
Don't forget to add your task estimate here.

Answers on page 114.

Do your tasks add up?

Did you notice a possible problem with your estimates? We've got a user story with an estimate, but now we're adding *new* estimates to our tasks. What happens when the two sets of estimates don't agree?


Title: Order flowers
Est: 8 days
Priority:
Your user story's estimate
"Order Flowers" development tasks
Task 5
Task 6
Task 7
Create Send Flowers event that contains the address and flower order
2
3
+
4
+
2
8 days
?
9 days
Both of these estimates can't be right...right?

Task estimates add confidence to user story estimates

Your user story estimates kept you in the right ballpark when you were planning your iterations, but tasks really add another level of detail specific to the actual coding you'll do for a user story.

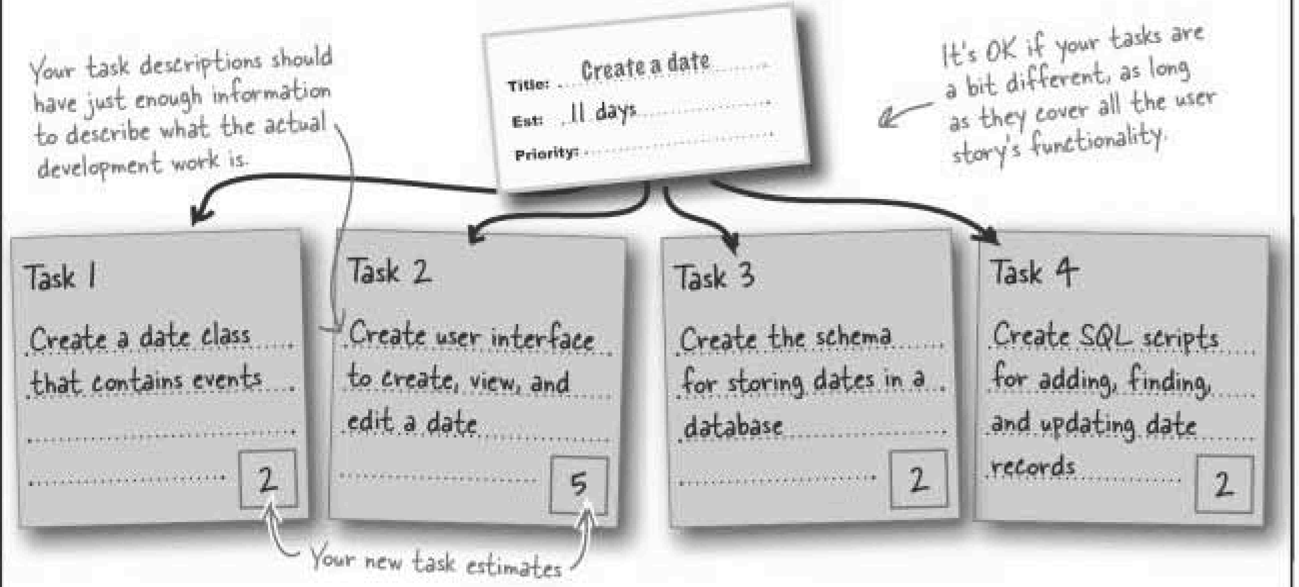
In fact, it's often best to break out tasks from your user stories right ***at the beginning*** of the estimation process, if you have time. This way you'll add even more confidence to the plan that you give your customer. ***It's always best to rely on the task estimates.*** Tasks describe the actual software development work that needs to be done and are far less of a guesstimate than a coarse-grained user story estimate.

Break user stories into tasks to add CONFIDENCE to your estimates and your plan.

And the earlier you can do this, the better.

Sharpen your pencil Solution

You were asked to take the user story of creating a date and break out the tasks that you think you and your team will need to execute to develop this user story, not forgetting to add task estimates...


Title: Create a date
Est: 11 days
Priority:
Your task descriptions should have just enough information to describe what the actual development work is.
It's OK if your tasks are a bit different, as long as they cover all the user story's functionality.
Task 1
Create a date class that contains events
2
Task 2
Create user interface to create, view, and edit a date
5
Task 3
Create the schema for storing dates in a database
2
Task 4
Create SQL scripts for adding, finding, and updating date records
2
Your new task estimates

there are no Dumb Questions

Q: My tasks add up to a new estimate for my user story, so were my original user story estimates wrong?

A: Well, yes and no. Your user story estimate was accurate enough in the beginning to let you organize your iterations. Now, with task estimates, you have a set of **more accurate data** that either backs up your user story estimates or conflicts with them.

You always want to rely on data that you trust, the estimates that you feel are most accurate. In this case, those are your task estimates.

Q: How big should a task estimate be?

A: Your task estimates should ideally be between 1/2 and 5 days in length. A shorter task, measured in hours, is too small a task. A task that is longer than five days spreads across more than one working week, and that gives the developer working on the task too much time to lose focus.

Q: What happens when I discover a big missing task?

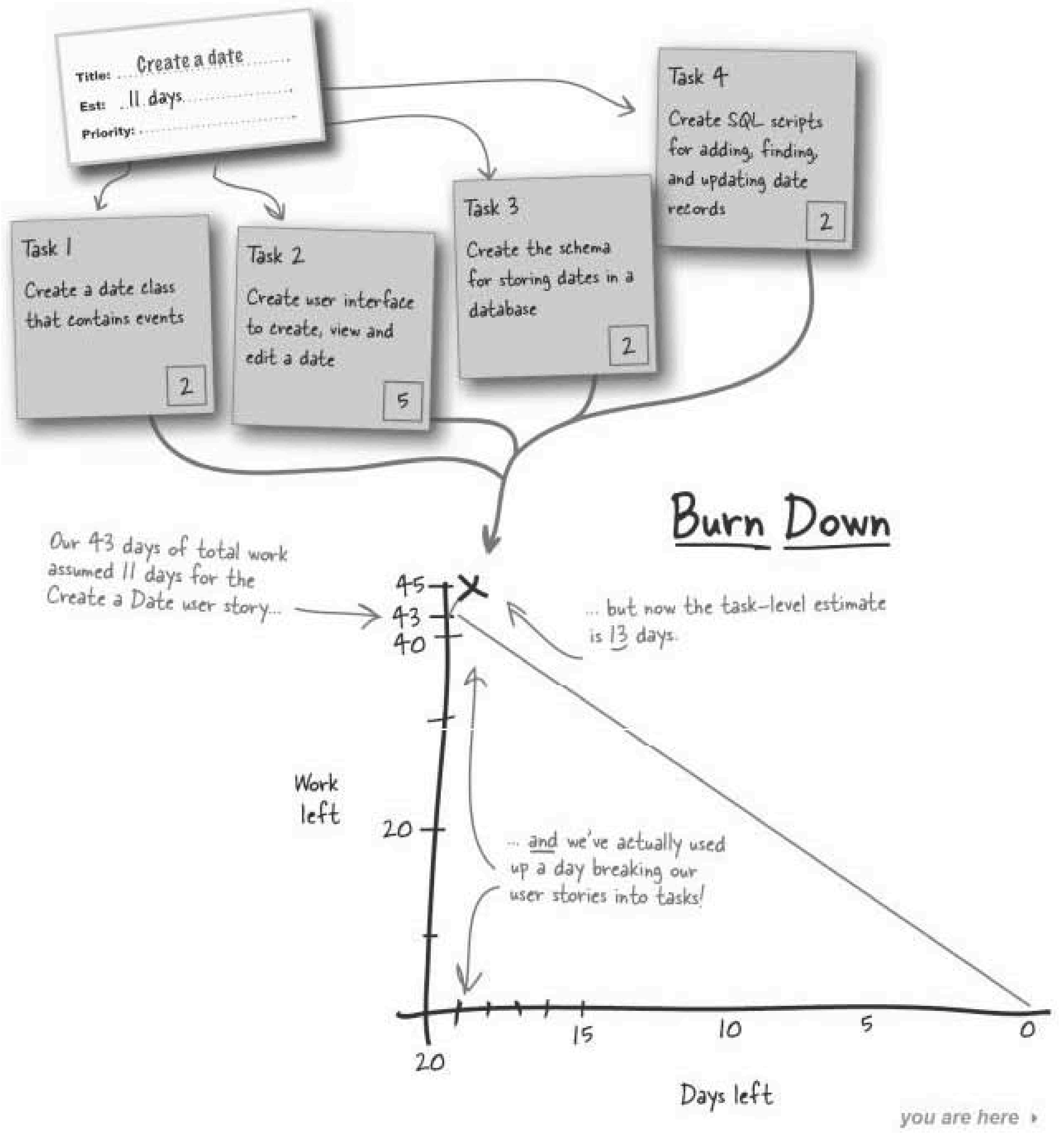
A: Sometimes—hopefully not too often—you'll come across a task that just breaks your user story estimate completely. You might have forgotten something important when first coming up with the user story estimates, and suddenly the devil in the details rears its ugly head, and you have a more accurate, task-based estimate that completely blows your user story estimate out of the water.

When this happens you can really only do one thing, and that's adjust your iteration. To keep your iteration within 20 working days, you can postpone that large task (and user story) until the next iteration, reshuffling the rest of your iterations accordingly.

To avoid these problem, you could break your user stories into tasks earlier. For instance, you might break up your user stories into tasks when you initially plan your iterations, always relying on your task estimates over your original user story estimates as you balance out your iterations to 20 working days each.

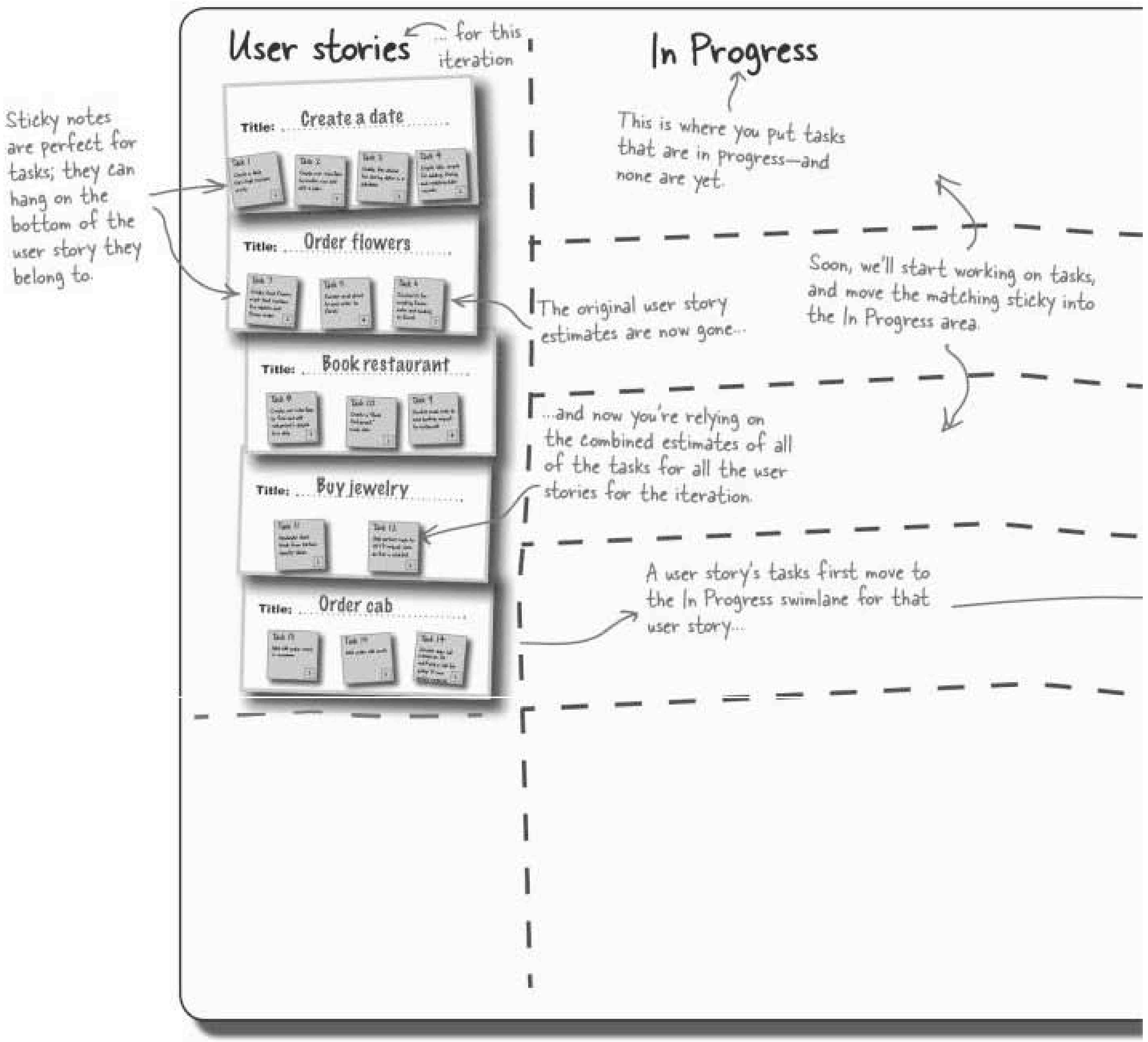
Plot just the work you have left

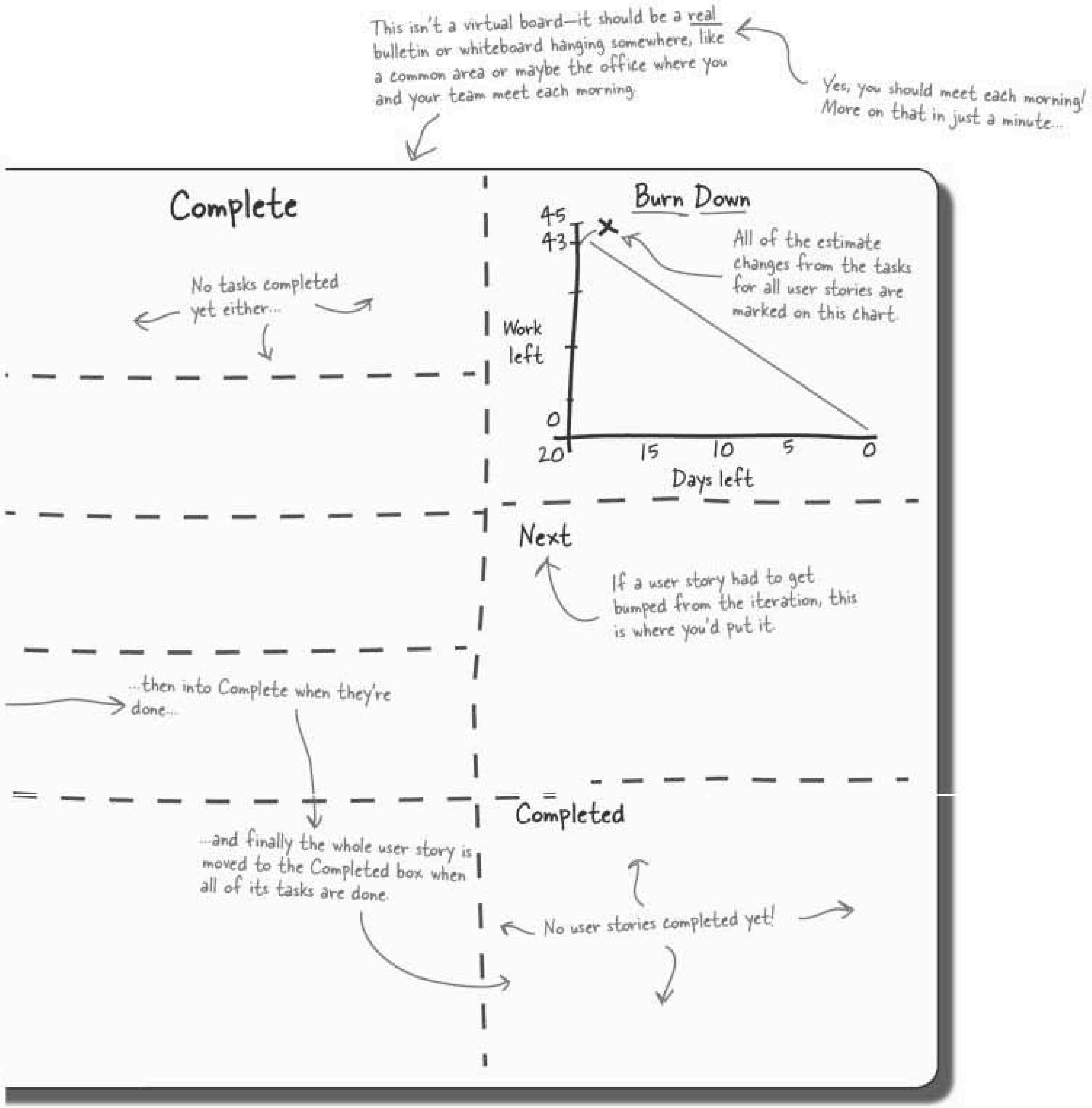
Remember that burn-down rate chart from Chapter 3? Here's where it starts to help us track what's going on in our project. Every time we do any work or review an estimate, we update our new estimates, and the time we have left, on our burn-down chart:


Title: Create a date
Est: 11 days
Priority:
Task 1
Create a date class that contains events
2
Task 2
Create user interface to create, view and edit a date
5
Task 3
Create the schema for storing dates in a database
2
Task 4
Create SQL scripts for adding, finding, and updating date records
2
Burn Down
Our 43 days of total work assumed 11 days for the Create a Date user story...
... but now the task-level estimate is 13 days.
... and we've actually used up a day breaking our user stories into tasks!
45
43
40
20
Work left
20
15
10
5
0
Days left

Add your tasks to your board

You and your team are now almost ready to start working on your tasks, but first you need to update the big board on your wall. Add your task sticky notes to your user stories. and also add an In Progress and Complete section for tracking tasks and user stories:

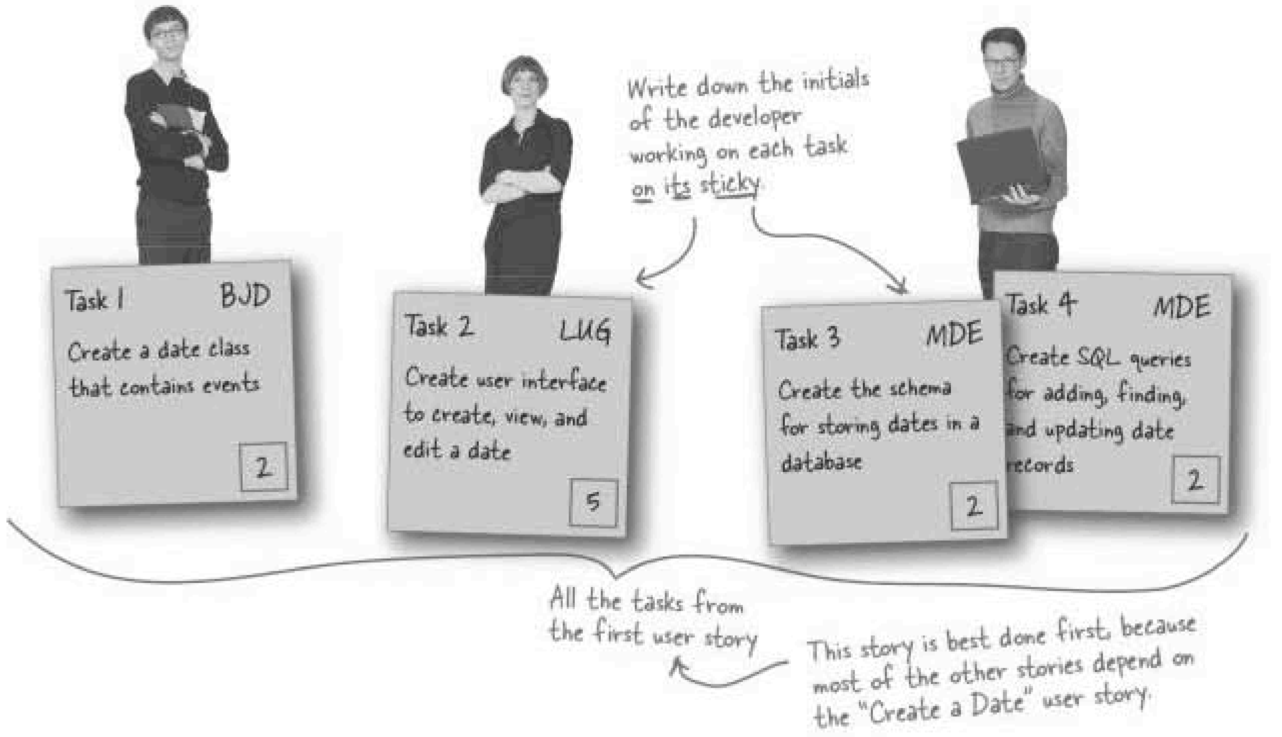

User stories
... for this iteration
In Progress
This is where you put tasks that are in progress—and none are yet.
Sticky notes are perfect for tasks; they can hang on the bottom of the user story they belong to.
Title: Create a date
Title: Order flowers
Title: Book restaurant
Title: Buy jewelry
Title: Order cab
The original user story estimates are now gone...
...and now you're relying on the combined estimates of all of the tasks for all the user stories for the iteration.
Soon, we'll start working on tasks, and move the matching sticky into the In Progress area.
A user story's tasks first move to the In Progress swimlane for that user story...


This isn't a virtual board—it should be a real bulletin or whiteboard hanging somewhere, like a common area or maybe the office where you and your team meet each morning.
Yes, you should meet each morning! More on that in just a minute...
Complete
No tasks completed yet either...
Burn Down
45
43
Work left
0
20
15
10
5
0
Days left
All of the estimate changes from the tasks for all user stories are marked on this chart.
Next
If a user story had to get bumped from the iteration, this is where you'd put it.
...then into Complete when they're done...
Completed
...and finally the whole user story is moved to the Completed box when all of its tasks are done.
No user stories completed yet!

Start working on your tasks

It's time to bring that burn-down rate back under control by getting started developing on your first user story. And, with small tasks, you can assign your team work in a sensible, trackable way:

Title: Create a date
Est: 11 days
Priority:


Write down the initials of the developer working on each task on its sticky.
Task 1 BJD
Create a date class that contains events
2
Task 2 LUG
Create user interface to create, view, and edit a date
5
Task 3 MDE
Create the schema for storing dates in a database
2
Task 4 MDE
Create SQL queries for adding, finding, and updating date records
2
All the tasks from the first user story
This story is best done first, because most of the other stories depend on the "Create a Date" user story.

there are no Dumb Questions

Q: How do I figure out who to assign a task to?

A: There are no hard-and-fast rules about who to give a task to, but it's best to just apply some common sense. Figure out who would be most productive or—if you have the time, will learn most from a particular task by looking at their own expertise—and then allocate the task to the best-suited developer, or the one who will gain the most, that's not already busy.

Q: Why allocate tasks just from the first user story. Why not take one task from each user story?

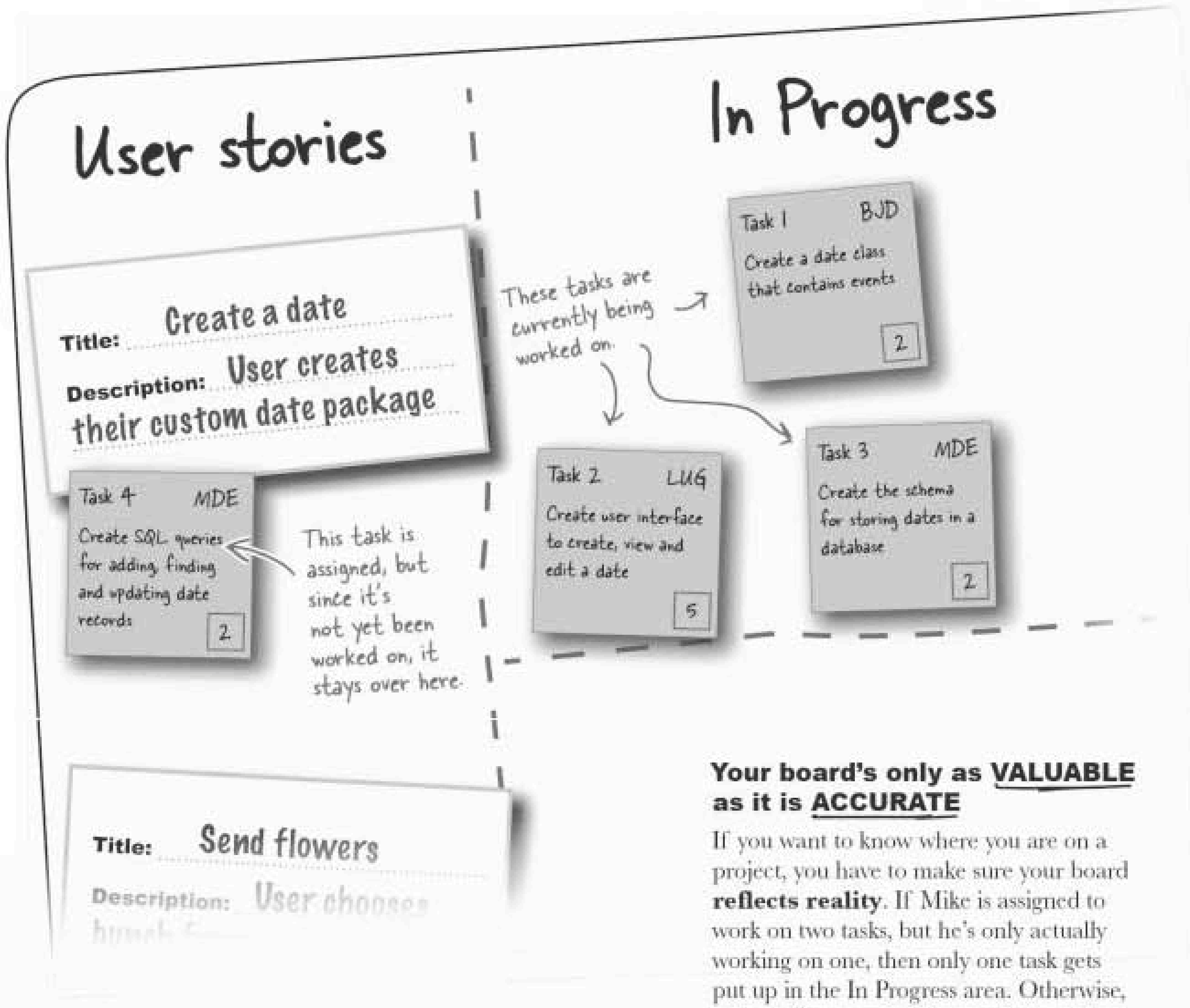
A: One good reason is so that so that you don't wind up with five stories in a half-done state, and instead can wrap up a user story and move on to the next. If you've got one story your other stories depend on, you may want to get all that first story's tasks done at once. However, if your stories are independent of each other, you may work on tasks from multiple stories all at the same time.

Q: I'm still worried about that burn-down rate being way up, is there anything I can do right now to fix that?

A: A burn-down rate that's going up is always a cause for concern, but since you're early on, let's wait a bit and see if we catch up.

A task is only in progress when it's IN PROGRESS

Now that everyone's got some work to do, it's time to move those task stickies off of user story cards, and onto the In Progress area of your big board. But you only put tasks that are **actually being worked on** in the In Progress column—even if you already know who'll be working on tasks yet to be tackled.

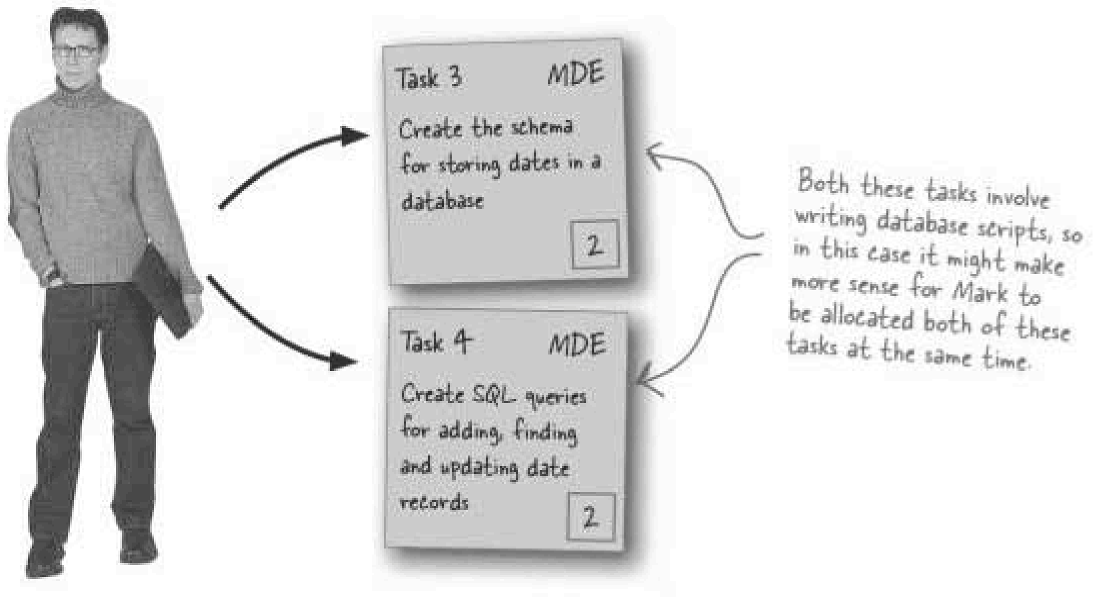

User stories
In Progress
Title: Create a date
Description: User creates their custom date package
Task 4 MDE
Create SQL queries for adding, finding and updating date records
2
This task is assigned, but since it's not yet been worked on, it stays over here.
These tasks are currently being worked on.
Task 1 BJD
Create a date class that contains events
2
Task 2 LUG
Create user interface to create, view and edit a date
5
Task 3 MDE
Create the schema for storing dates in a database
2
Title: Send flowers
Description: User chooses

Your board's only as VALUABLE as it is ACCURATE

If you want to know where you are on a project, you have to make sure your board **reflects reality**. If Mike is assigned to work on two tasks, but he's only actually working on one, then only one task gets put up in the In Progress area. Otherwise, it looks like more is being worked on than really is.

What if I'm working on two things at once?

Not all tasks are best executed in isolation. Sometimes two tasks are related, and, because there is so much overlap, it's actually more work to tackle one, and then the other separately. In these cases the most productive thing to do is work on those tasks ***at the same time***...


Task 3
MDE
Create the schema for storing dates in a database
2
Task 4
MDE
Create SQL queries for adding, finding and updating date records
2
Both these tasks involve writing database scripts, so in this case it might make more sense for Mark to be allocated both of these tasks at the same time.

Sometimes working on both tasks at the same time IS the best option

When you have two tasks that are closely related. then it's not really a problem to work on them both at the same time.

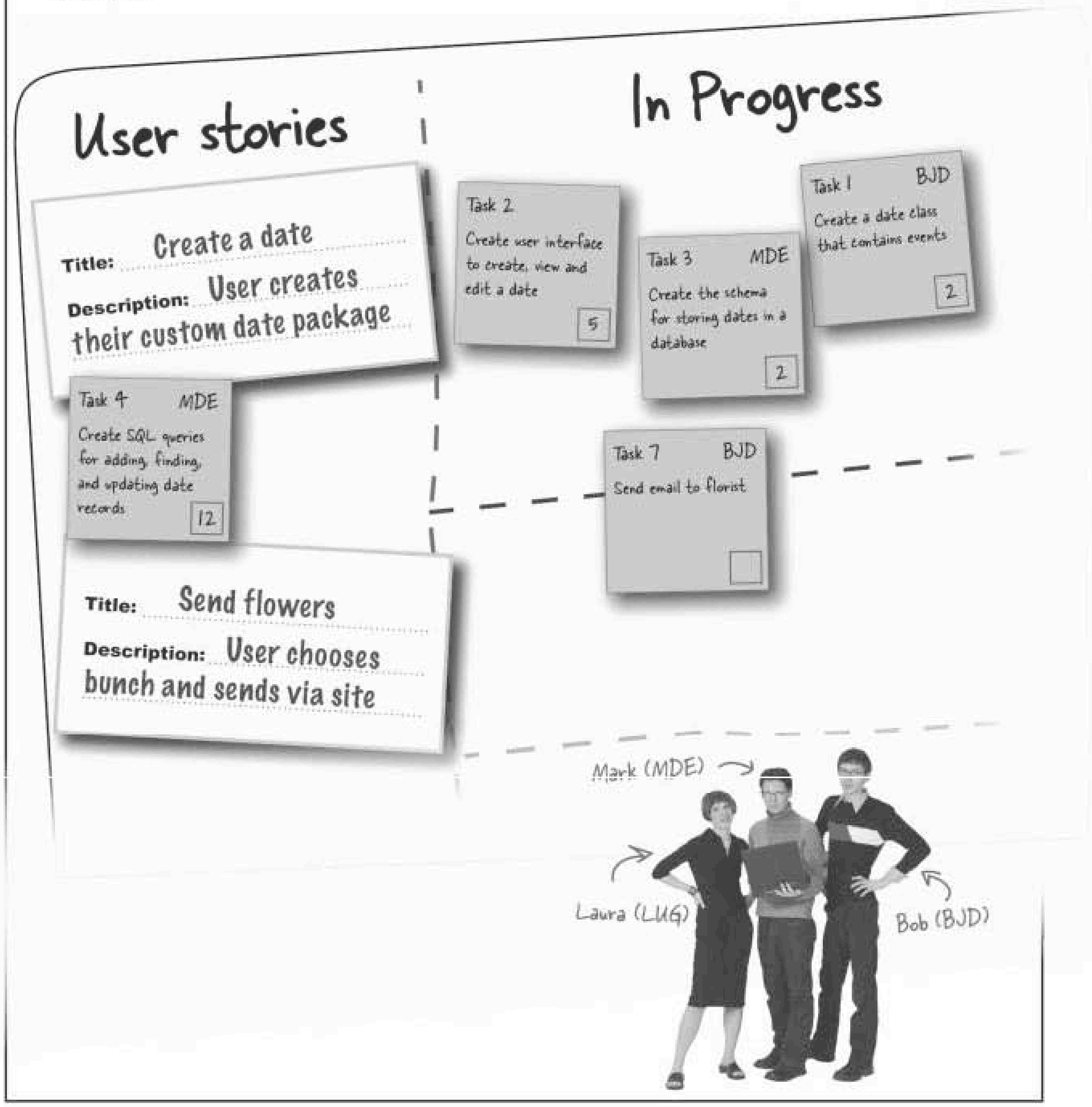
This is especially the case where the work completed in one task could ***inform decisions*** made in the work for another task. Rather than completing one task and starting the next, and then realizing that you need to do some work on the first task again, it is far more efficient to work both tasks at the same time.

Rules of Thumb

- Try to double-up tasks that are related to each other, or at least focus on roughly the same area of your software. The less thought involved in moving from one task to another, the faster that switch will be.
- Try not to double-up on tasks that have large estimates. It's not only difficult to stay focused on a long task, but you will be more confident estimating the work involved the shorter the task is.

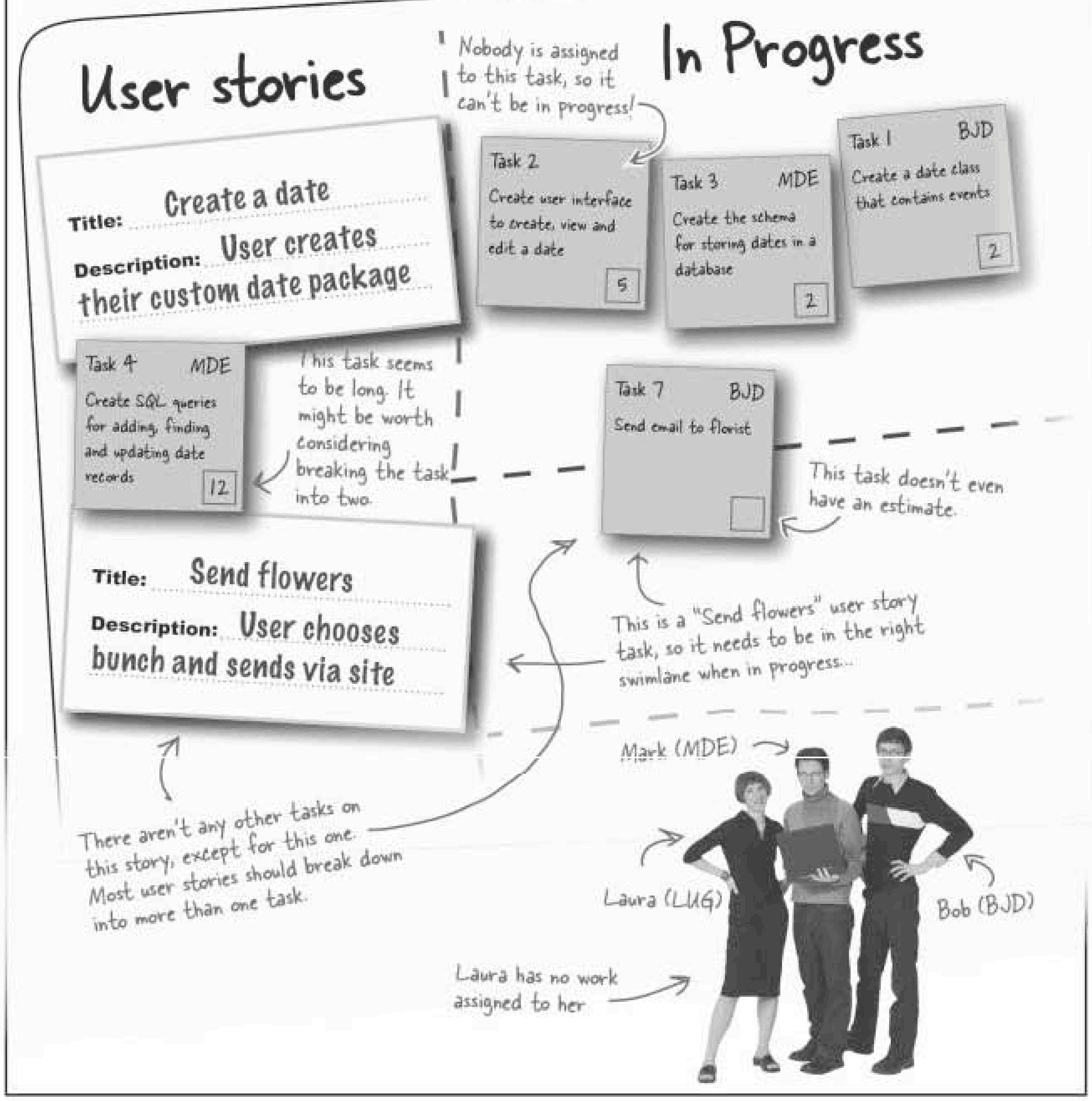
Exercise

Someone's been tampering with the board and things are a real mess. Take a look at the project below and annotate all of the problems you can spot.


User stories
In Progress
Title: Create a date
Description: User creates their custom date package
Task 2
Create user interface to create, view and edit a date
5
Task 3 MDE
Create the schema for storing dates in a database
2
Task 1 BJD
Create a date class that contains events
2
Task 4 MDE
Create SQL queries for adding, finding, and updating date records
12
Task 7 BJD
Send email to florist
Title: Send flowers
Description: User chooses bunch and sends via site
Mark (MDE)
Laura (LUG)
Bob (BJD)

Exercise Solution

Your job was to to take a look at the project below and annotate all of the problems you could spot...


User stories
In Progress
Title: Create a date
Description: User creates their custom date package
Task 2
Create user interface to create, view and edit a date
5
Task 3 MDE
Create the schema for storing dates in a database
2
Task 1 BJD
Create a date class that contains events
2
Task 4 MDE
Create SQL queries for adding, finding and updating date records
12
Task 7 BJD
Send email to florist
Title: Send flowers
Description: User chooses bunch and sends via site
Mark (MDE)
Laura (LUG)
Bob (BJD)

Nobody is assigned to this task, so it can't be in progress!

This task seems to be long. It might be worth considering breaking the task into two.

This task doesn't even have an estimate.

This is a "Send flowers" user story task, so it needs to be in the right swimlane when in progress...

There aren't any other tasks on this story, except for this one. Most user stories should break down into more than one task.

Laura has no work assigned to her

Your first standup meeting...

A meeting so quick you don't even have time to sit down.

You've now got some tasks in progress, and so to keep everyone in the loop, while not taking up too much of their time, you conduct a quick standup meeting every day.

G'morning everyone, it's day 1 and I thought I'd call a quick meeting so we can update the board and get everything set for today's development...

Mark: So, we've all had our tasks for one day now. How are we doing?

Bob: Well, I haven't hit any big problems yet, so nothing new really to report.

Mark: That's great. I've had a bit of success and finished up on the scripts to create tables in the database...

Laura: Things are still in progress on my user interface task.

Mark: OK, that all sounds good, I'll update the board and move my task into Completed. We can update the burn rate, too; maybe we're making up for some of that time we lost earlier. Any other successes or issues to report?

Bob: Well, I guess I should probably mention that I'm finding creating the right Date class a little tricky...

Mark: That's fine. I'm really glad you brought it up, though. That's a two-day task and we need it done tomorrow, so I'll get you some help on that as soon as possible. OK, it's been about seven minutes, I think we're done here...

Your daily standup meetings should:

- **Track your progress.** Get everyone's input about how things are going.
- **Update your burn-down rate.** It's a new day so you need to update your burndown rate to see how things are going.
- **Update tasks.** If a task is completed then it's time to move it over into the Completed area and check those days off of your burn-down rate.
- **Talk about what happened yesterday and what's going to happen today.** Bring up any successes that happened since yesterday's standup meeting and make sure everyone knows what they're doing today.
- **Bring up any issues.** The standup meeting is not a place to be shy, so encourage everyone to bring up any problems they've encountered so that you all as a team can start to fix those problems.
- **Last between 5 and 15 minutes.** Keep things brief and focused on the short-term tasks at hand.

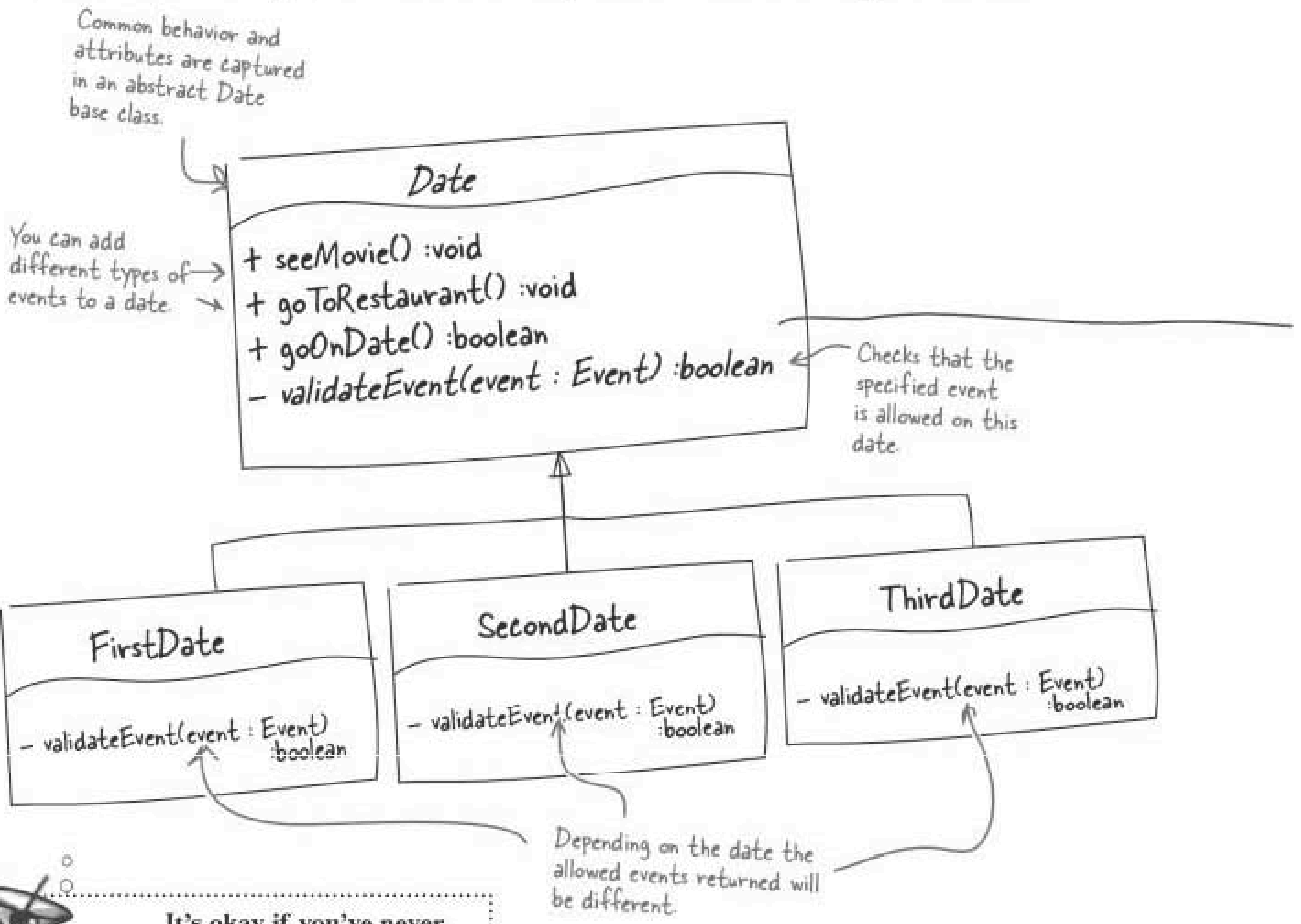
A daily standup meeting should keep everyone motivated, keep your board up-to-date, and highlight any problems early.

Task 1: Create the Date class

Bob's been busy creating the classes that bring the "Create a Date" user story to life, but he needs a hand. Here's a UML class diagram that describes the design he's come up with so far.

A UML class diagram shows the classes in your software and how they relate to each other.

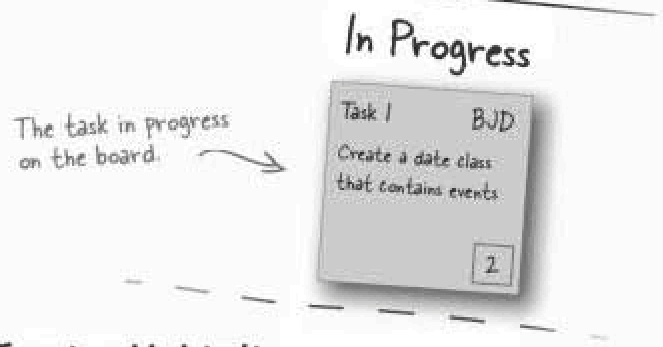
The Date class is split into three classes, one class for each type of date...



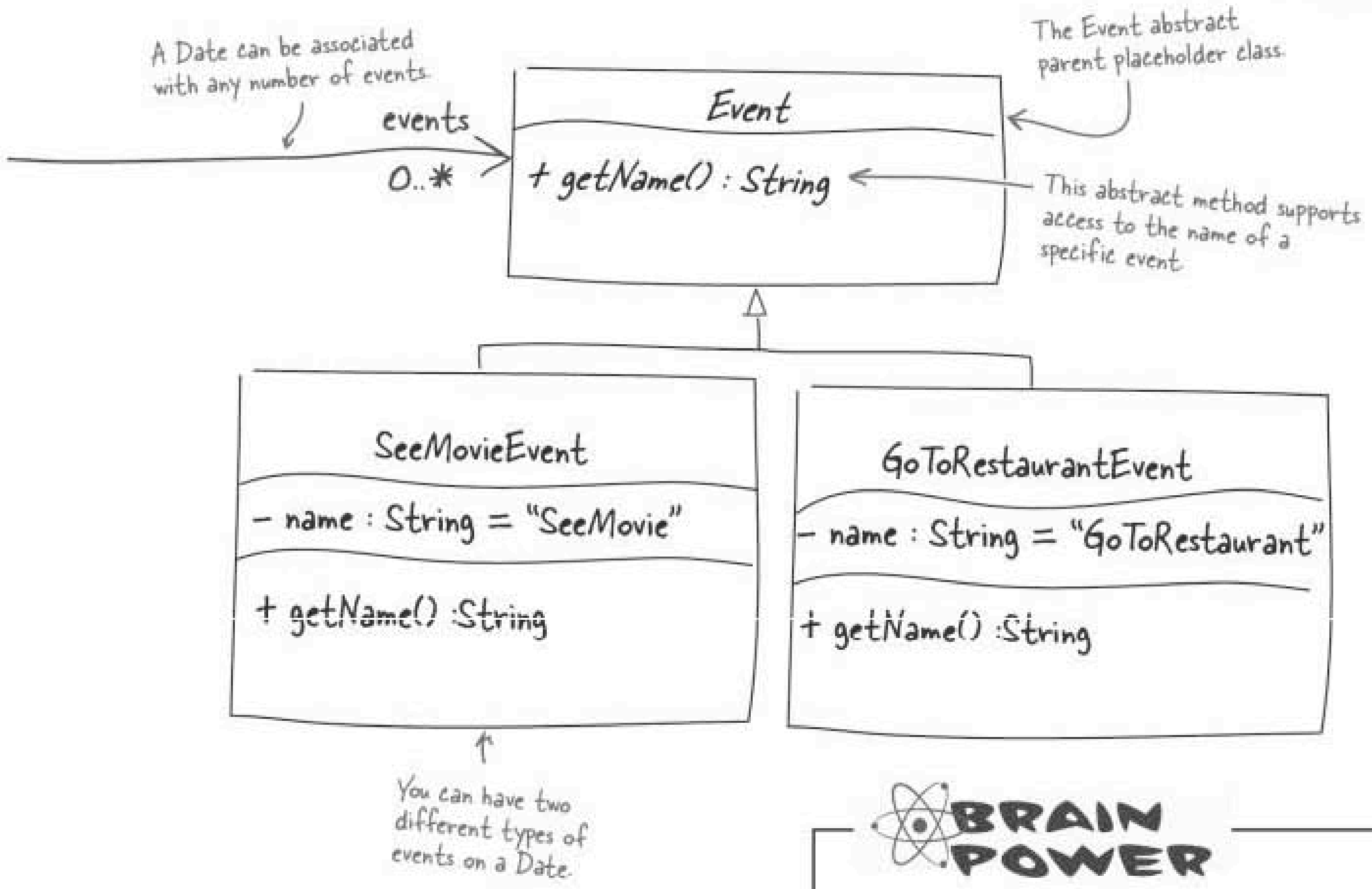
Common behavior and attributes are captured in an abstract Date base class.
Date
You can add different types of events to a date.
+ seeMovie() :void
+ goToRestaurant() :void
+ goOnDate() :boolean
– validateEvent(event : Event) :boolean
Checks that the specified event is allowed on this date.
FirstDate
– validateEvent(event : Event) :boolean
SecondDate
– validateEvent(event : Event) :boolean
ThirdDate
– validateEvent(event : Event) :boolean
Depending on the date the allowed events returned will be different.

It's okay if you've never seen UML before!

Don't worry if you don't know your UML class diagrams from your sequences; there's a short overview in Appendix i to help you get comfortable with UML notation as quickly as possible.


In Progress
The task in progress on the board.
Task 1
BJD
Create a date class that contains events
2

Each Date can then have a number of Events added to it...


A Date can be associated with any number of events.
events
0..*
Event
+ getName() : String
The Event abstract parent placeholder class.
This abstract method supports access to the name of a specific event.
SeeMovieEvent
- name : String = "SeeMovie"
+ getName() :String
GoToRestaurantEvent
- name : String = "GoToRestaurant"
+ getName() :String
You can have two different types of events on a Date.

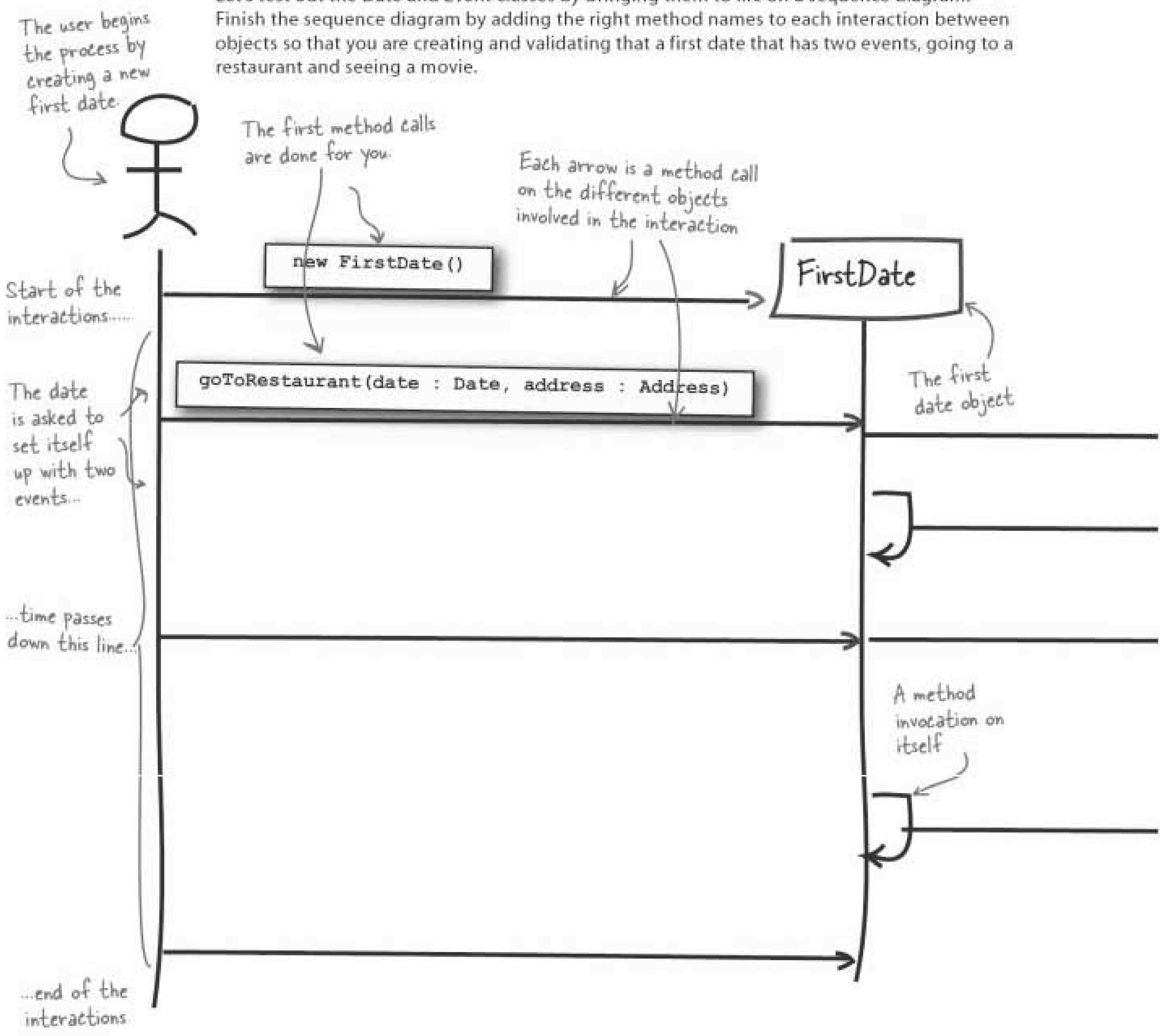
BRAIN POWER

What do you think of this design?

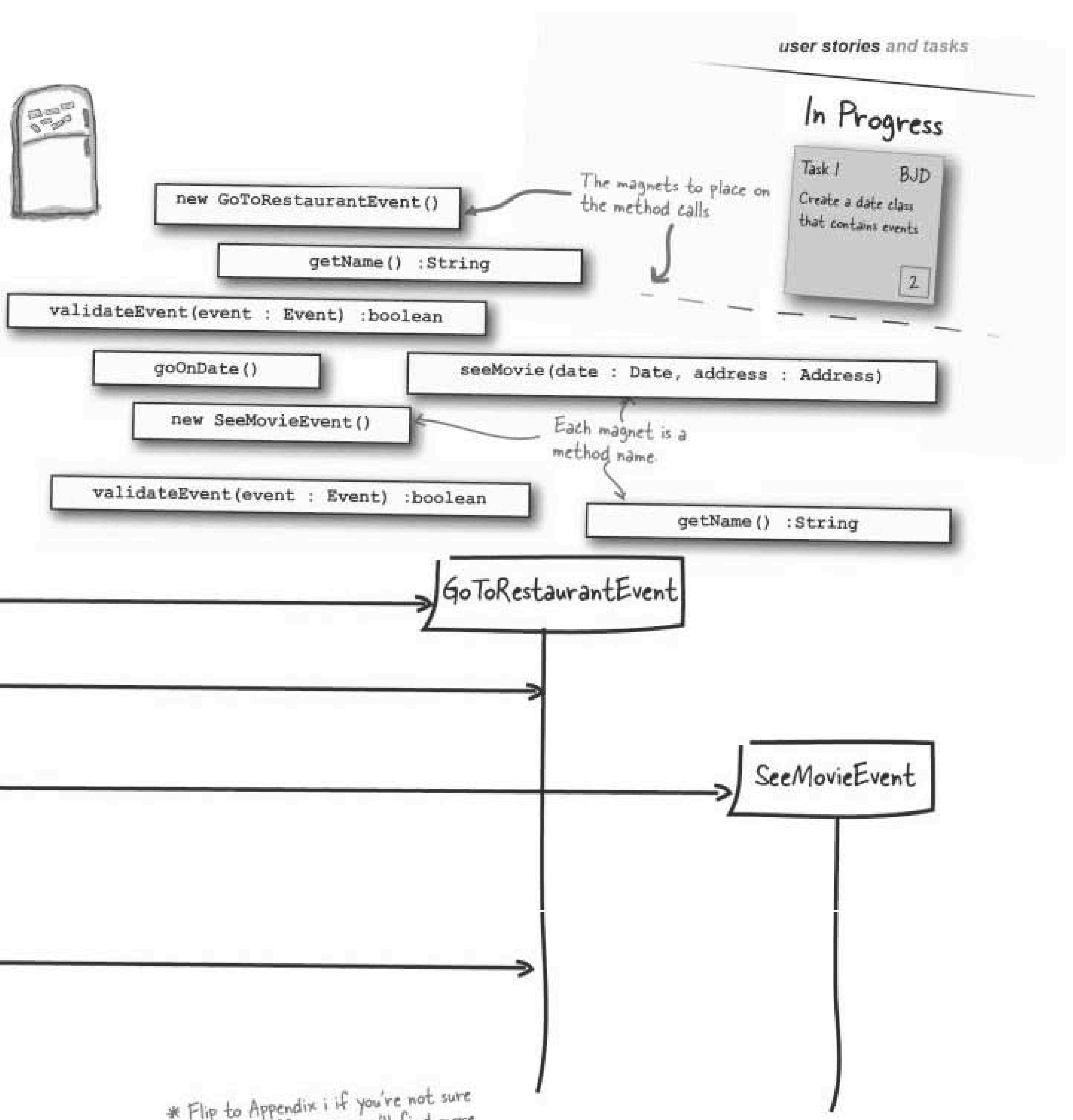
Task 1: Creating dates

A diagram that brings objects to life, showing how they work together to make an interaction happen

Let's test out the Date and Event classes by bringing them to life on a sequence diagram. Finish the sequence diagram by adding the right method names to each interaction between objects so that you are creating and validating that a first date that has two events, going to a restaurant and seeing a movie.

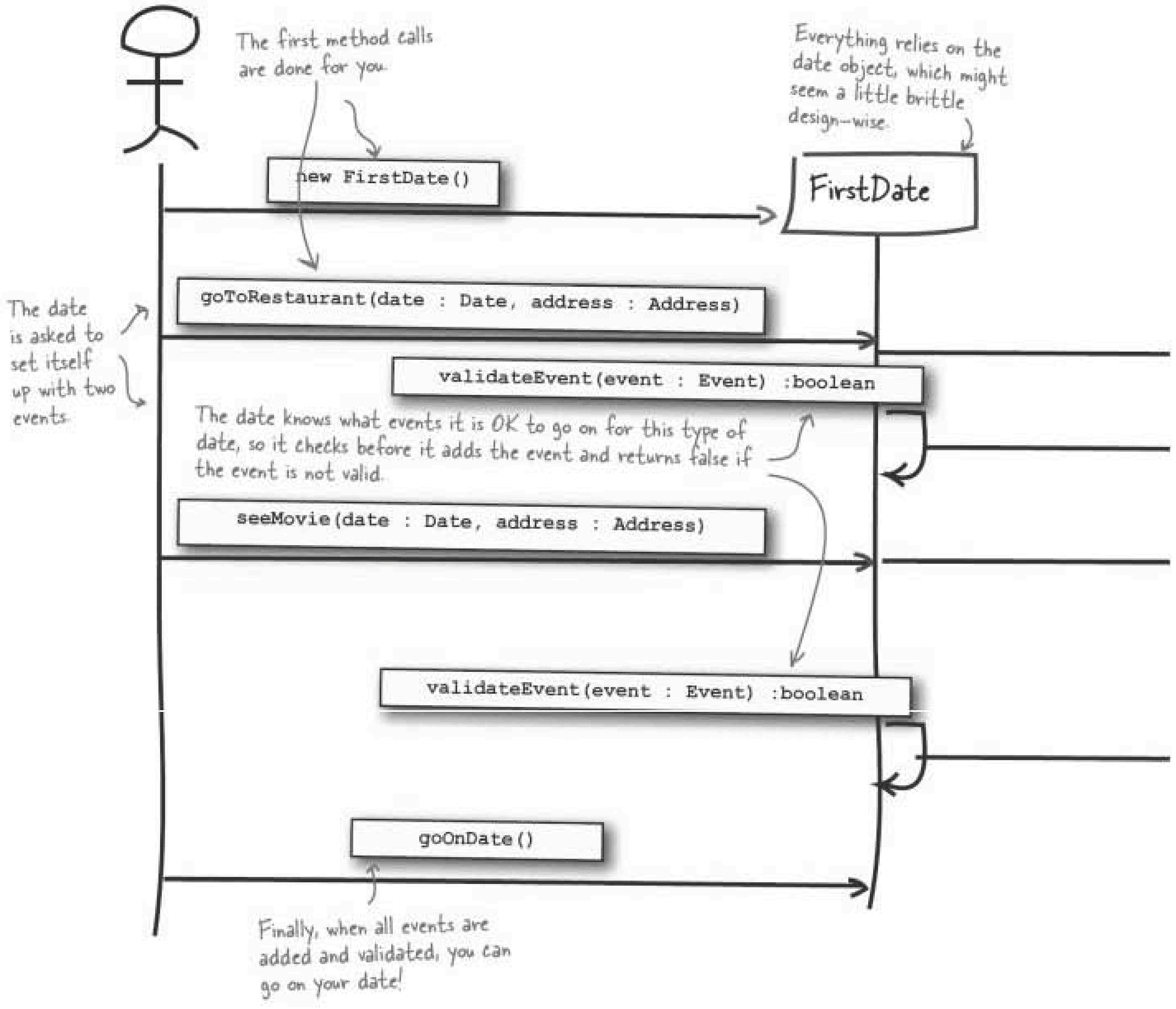


The user begins the process by creating a new first date.
The first method calls are done for you.
Each arrow is a method call on the different objects involved in the interaction
new FirstDate()
FirstDate
Start of the interactions.....
goToRestaurant(date : Date, address : Address)
The first date object
The date is asked to set itself up with two events...
...time passes down this line...
A method invocation on itself
...end of the interactions


In Progress
Task 1
BJD
Create a date class that contains events
2
The magnets to place on the method calls
Each magnet is a method name
new GoToRestaurantEvent()
getName() :String
validateEvent(event : Event) :boolean
goOnDate()
seeMovie(date : Date, address : Address)
new SeeMovieEvent()
validateEvent(event : Event) :boolean
getName() :String
GoToRestaurantEvent
SeeMovieEvent

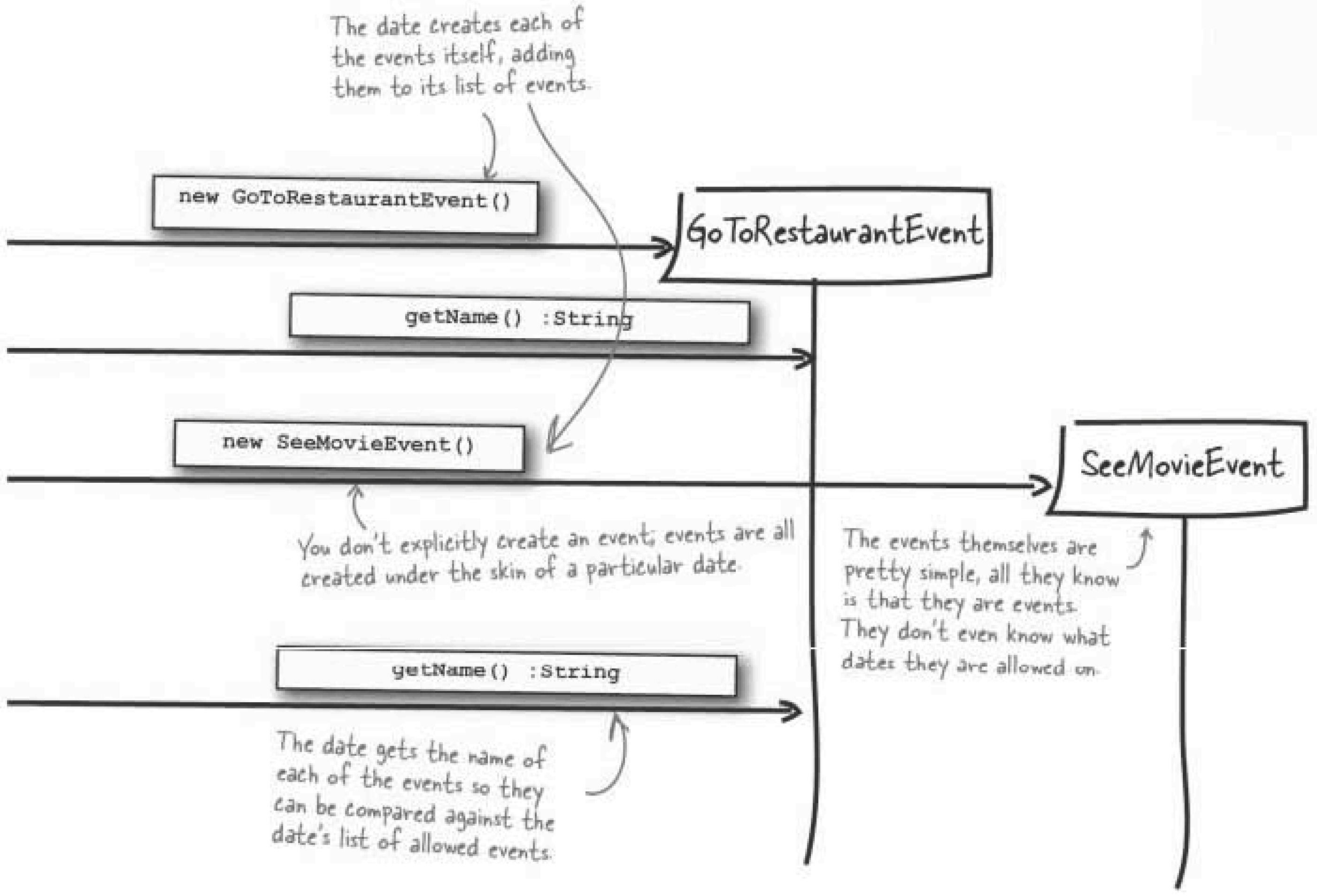
* Flip to Appendix i if you're not sure what this stuff means; you'll find more on UML class diagrams and sequence diagrams there.

Your job was to test out the Date and Event classes by bringing them to life on a sequence diagram. You should have finished the sequence diagram so that you plan and go on a first date with two events, going to a restaurant and seeing a movie.



The first method calls are done for you.
Everything relies on the date object, which might seem a little brittle design-wise.
new FirstDate()
FirstDate
goToRestaurant(date : Date, address : Address)
The date is asked to set itself up with two events.
validateEvent(event : Event) :boolean
The date knows what events it is OK to go on for this type of date, so it checks before it adds the event and returns false if the event is not valid.
seeMovie(date : Date, address : Address)
validateEvent(event : Event) :boolean
goOnDate()
Finally, when all events are added and validated, you can go on your date!

In Progress
Task 1
BJD
Create a date class that contains events
2


The date creates each of the events itself, adding them to its list of events.
new GoToRestaurantEvent()
GoToRestaurantEvent
getName() :String
new SeeMovieEvent()
SeeMovieEvent
You don't explicitly create an event; events are all created under the skin of a particular date.
The events themselves are pretty simple, all they know is that they are events. They don't even know what dates they are allowed on.
getName() :String
The date gets the name of each of the events so they can be compared against the date's list of allowed events.

Standup meeting: Day 5, end of Week 1...

So, one day left in the first week, how are we doing according to the big board?

Bob: Well, I finally got the date class finished with a little help, ran late by a day though...

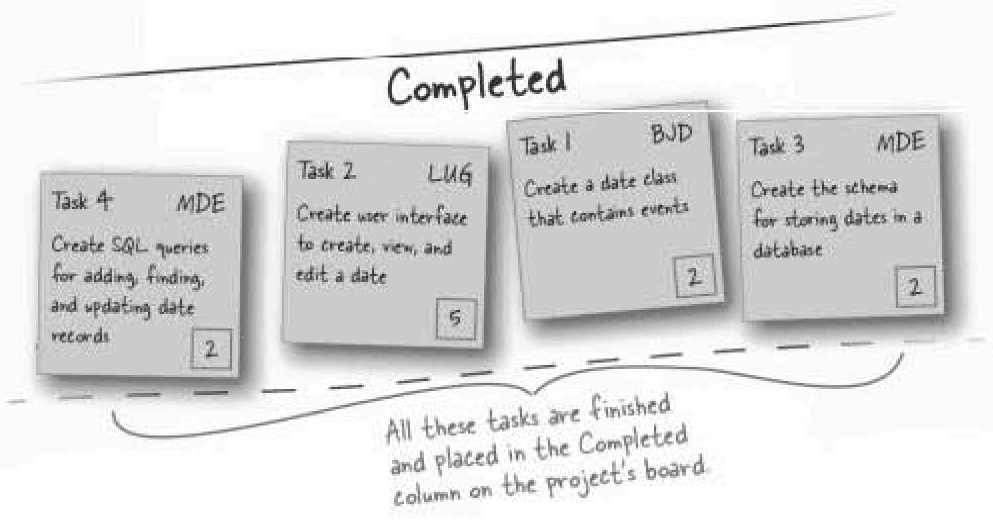
Laura: That's OK, this time around. We can hopefully make some of that time up later.

Mark: All work on the database is now done; I'm all set for the next set of tasks.

Laura: Great, and I got my work done on the user interface pieces, so we've actually got something running.

Bob: Always a good week when you head out of the office with something working...

Laura: Absolutely. OK, it's time to update the board and our burn-down rate to get things set up for next week.


Completed
Task 4 MDE Create SQL queries for adding, finding, and updating date records 2
Task 2 LUG Create user interface to create, view, and edit a date 5
Task 1 BJD Create a date class that contains events 2
Task 3 MDE Create the schema for storing dates in a database 2
All these tasks are finished and placed in the Completed column on the project's board.

there are no Dumb Questions

Q: Do I REALLY have to get everyone to stand up during a standup meeting?

A: No, not really. A standup meeting is called "standup" because it is meant to be a fast meeting that lasts a **maximum** of 15 minutes; you should ideally be aiming for 5 minutes.

We've all been stuck in endless meetings where nothing gets done, so the idea with a standup meeting is to keep things so short you don't even have time to find chairs. This keeps the focus and the momentum on only two agenda items:

- Are there any issues?
- Have we finished anything?

With these issues addressed, you can update your project board and get on with the actual development work.

Q: An issue has come up in my standup meeting that is going to take some discussion to resolve. Is it OK to lengthen the standup meeting to an hour to solve these bigger problems?

A: Always try to keep a standup meeting to less than 15 minutes. If an issue turns out to be something that requires further discussion, then schedule another meeting ***specifically for that issue***. The standup meeting has highlighted the issue, and so it's done its job.

Q: Do standup meetings have to be daily?

A: It certainly helps to make your standup meetings daily. With the pace of modern software development, issues arise on almost a daily basis, so a quick 15 minutes with your team is essential to keeping your finger on the pulse of the project.

Q: Is it best to do a standup meeting in the morning or the afternoon?

A: Ideally, standup meetings should be first thing in the morning. The meeting sets everyone up for the day's tasks and gives you time to hit issues straight away.

Still, there may be situations when you can't all meet in the morning, especially if you have remote employees. In those cases, standup meetings should be conducted when the majority of your team begin their working day. This isn't ideal for everyone, but at least most people get the full benefit of early feedback from the meeting.

On rare occasions, you can split the standup meeting in two. You might do this if part of your team works in a completely different time zone. If you go with this approach, keeping your board updated is even more critical, as this is the place where everyone's status from the standup meeting is captured for all to see.

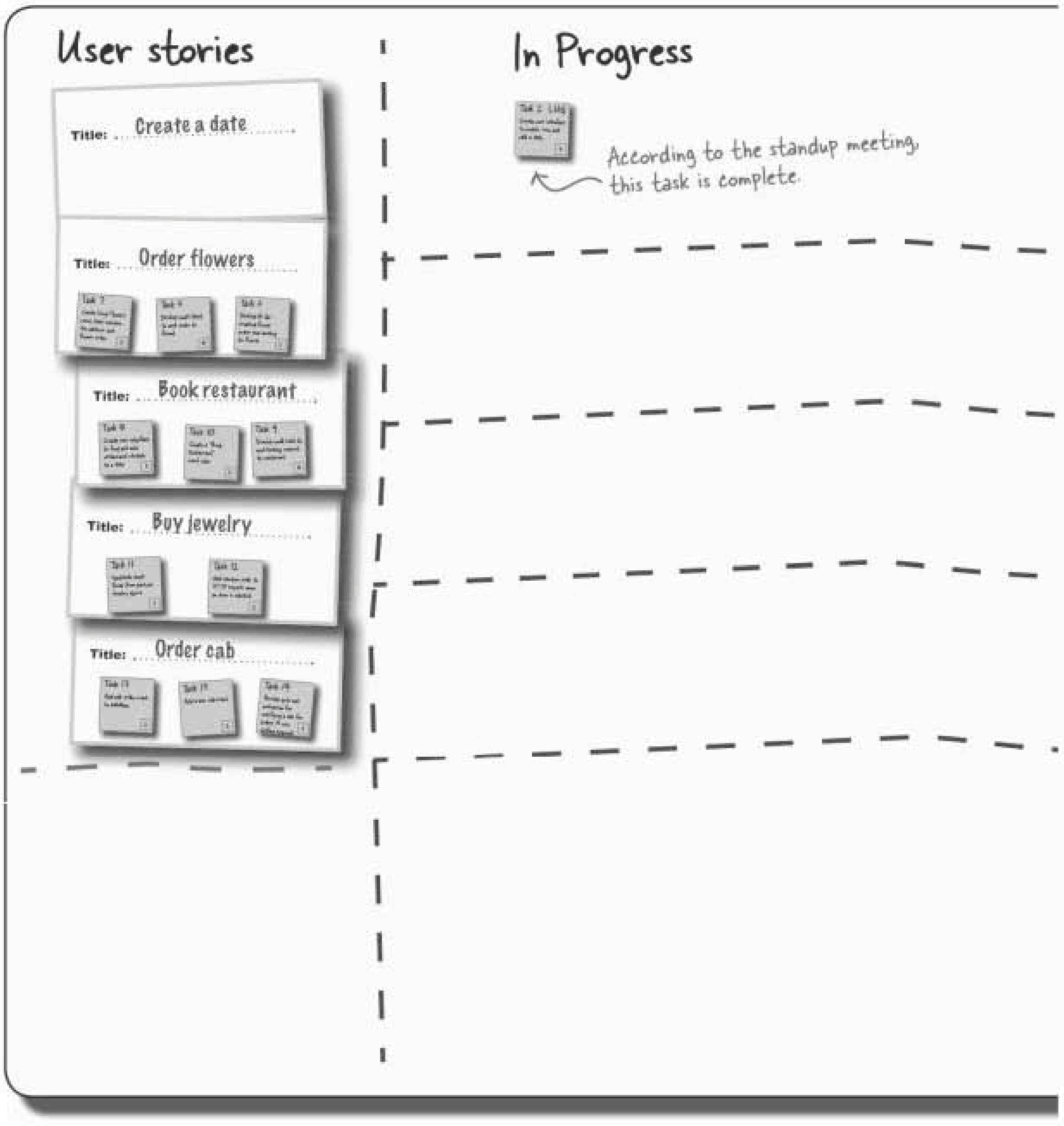
Standup meetings keep your peers, employees, and managers up to date, and keep your finger on the pulse of how your development work is going.

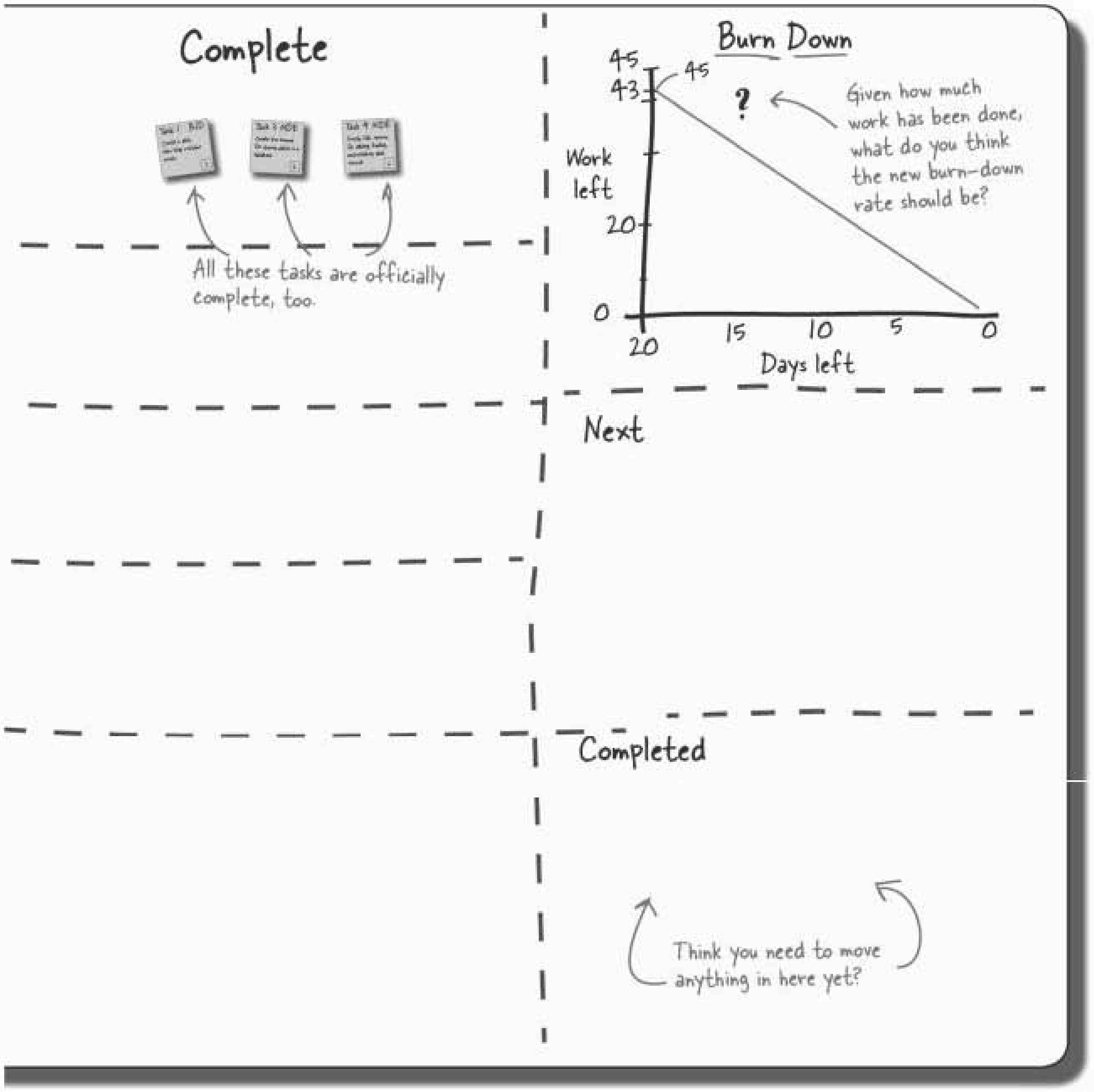
BULLET POINTS

- Organize **daily standup meetings** to make sure you catch issues early.
- Keep standup meetings **less than 15 minutes**.
- A standup meeting is all about **progress**, **problematic issues**, and **updating your board**.
- Try to schedule your standup meetings for the **morning** so that everyone knows where they are at the **beginning of the working day**.

Long Exercise

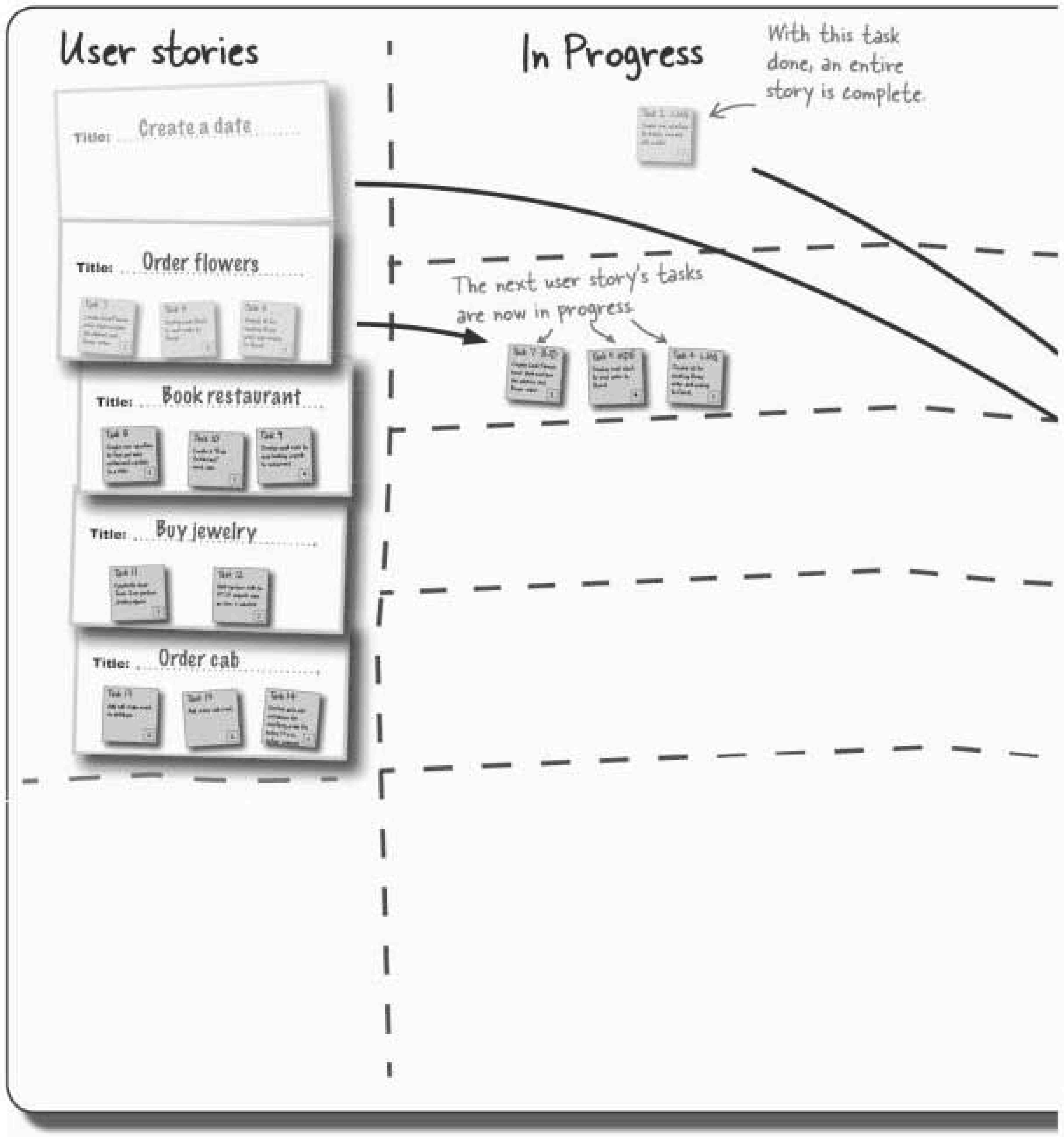
It's the end of Week 1, and you and the team have just finished your standup meeting. It's time to update the project board. Take a look at the board below and write down what you think needs to be changed and updated on the board to get it ready for Week 2.

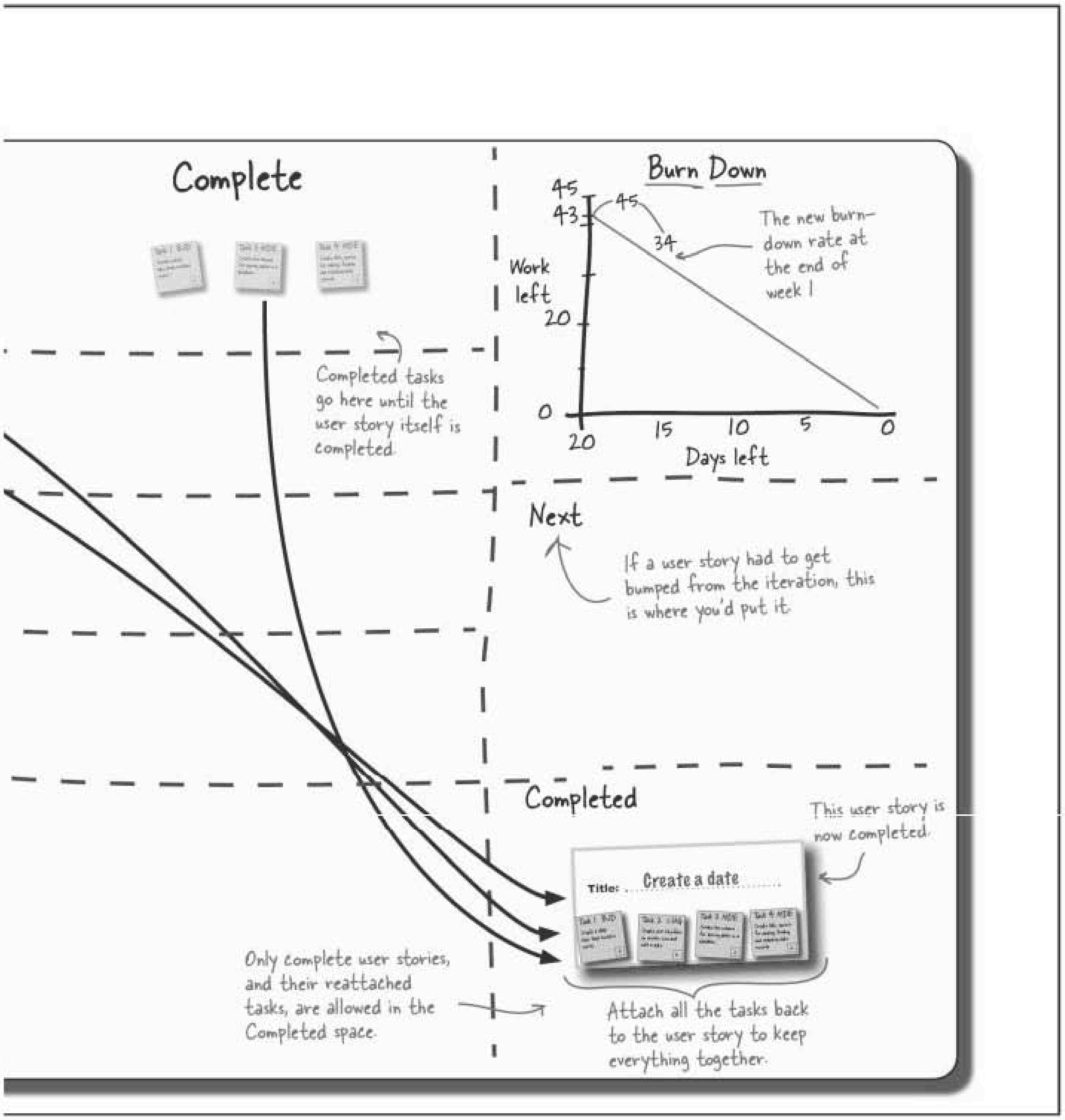



Complete
All these tasks are officially complete, too.
Burn Down
45
43
45
Work left
20
0
20
15
10
5
0
Days left
?
Given how much work has been done, what do you think the new burn-down rate should be?
Next
Completed
Think you need to move anything in here yet?

Long Exercise Solution

You were asked to update the board and write down what you think needs to be changed to get it ready for Week 2.




Complete
Completed tasks go here until the user story itself is completed.
Burn Down
45
43
45
34
20
0
Work left
20
15
10
5
0
Days left
The new burn-down rate at the end of week 1
Next
If a user story had to get bumped from the iteration, this is where you'd put it.
Completed
This user story is now completed.
Title: Create a date
Only complete user stories, and their reattached tasks, are allowed in the Completed space.
Attach all the tasks back to the user story to keep everything together.

Standup meeting: Day 2, Week 2...

In-Progress
Task 7
BJD
Create send flowers event that contains the address and flower order
3

One of the In Progres tasks from the board

Laura's acting as the team lead, at least on this iteration.

Hey guys, I've been busy working on my task and I noticed a way of saving us some time and effort by extending our design a little...

Laura: How are you going to do that?

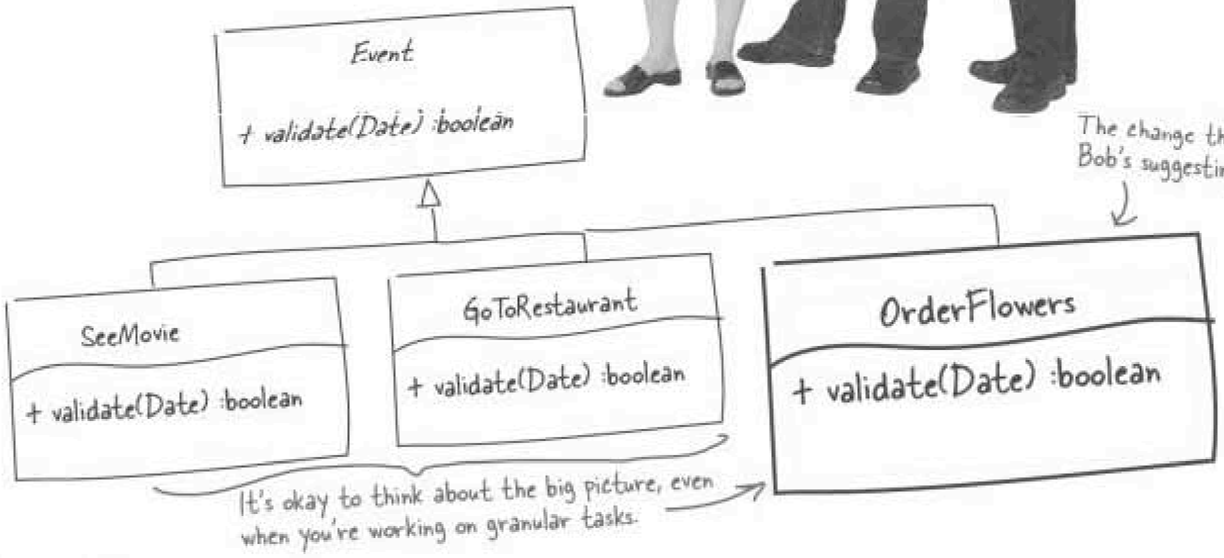
Bob: Well, if you treat someone ordering flowers as just another type of event, then we can add it straight into our current class tree, and that should save us some time in the long run.

Laura: That's sounds good. What do you think, Mark?

Mark: I don't see any problems right now...

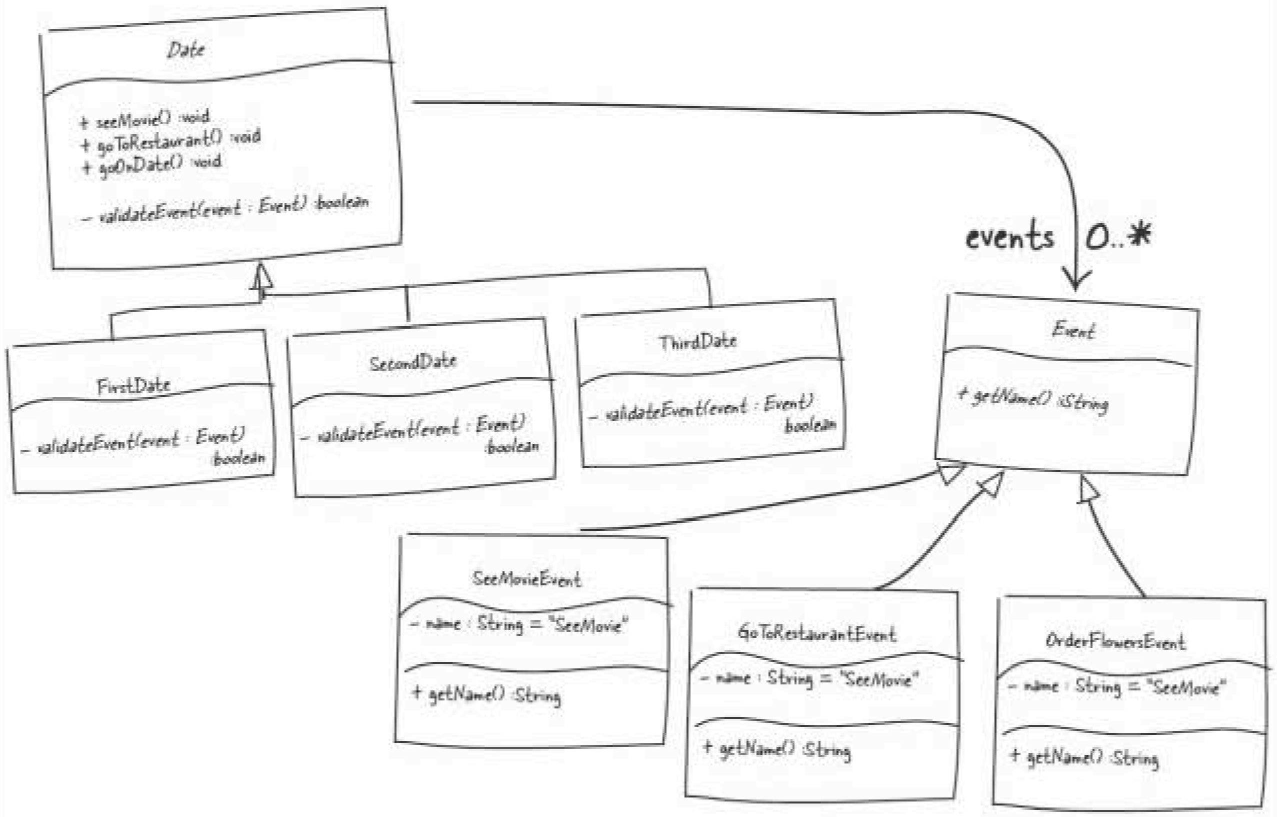
Bob: Apart from it might take an extra day right now to make the changes, but in the long run this should save us some time.

Laura: Mmm. We're still a little behind, but we can probably lose a day on the burn-down rate now if it saves us time later on in the iteration. OK, I'm sold, let's go for it...


Event
+ validate(Date) :boolean
SeeMovie
+ validate(Date) :boolean
GoToRestaurant
+ validate(Date) :boolean
OrderFlowers
+ validate(Date) :boolean
The change that Bob's suggesting
It's okay to think about the big picture, even when you're working on granular tasks.

Exercise

What refactoring do you think Bob is talking about? Take the class hierarchy below and circle all the things that you think will need to change to accommodate a new OrderFlowers event.


Date
+ seeMovie() :void
+ goToRestaurant() :void
+ goOnDate() :void
– validateEvent(event : Event) :boolean
events 0..*
Event
+ getName() :String
FirstDate
– validateEvent(event : Event) :boolean
SecondDate
– validateEvent(event : Event) :boolean
ThirdDate
– validateEvent(event : Event) :boolean
SeeMovieEvent
– name : String = "SeeMovie"
+ getName() :String
GoToRestaurantEvent
– name : String = "SeeMovie"
+ getName() :String
OrderFlowersEvent
– name : String = "SeeMovie"
+ getName() :String

How many classes did you have to touch to make Bob's changes? ....

....

....

....

Are you happy with this design? Why or why not?

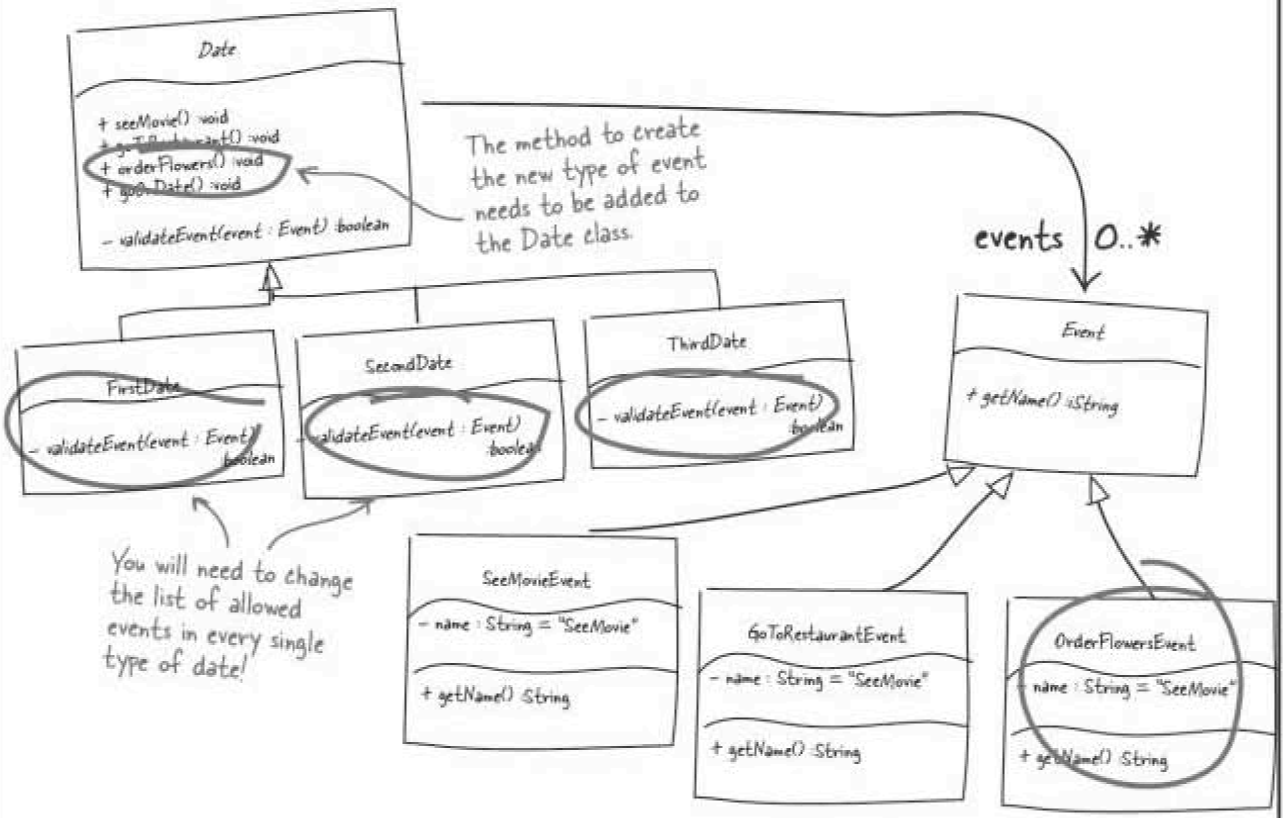
....

....

....

Exercise Solution

You were asked to take the class hierarchy below and circle all the places that you think will need to change to accommodate a new OrderFlowersEvent...


Date
+ seeMovie() :void
+ goToRestaurant() :void
+ orderFlowers() :void
+ goOnDate() :void
– validateEvent(event : Event) :boolean
The method to create the new type of event needs to be added to the Date class.
events 0..*
Event
+ getName() :String
FirstDate
– validateEvent(event : Event) :boolean
SecondDate
– validateEvent(event : Event) :boolean
ThirdDate
– validateEvent(event : Event) :boolean
You will need to change the list of allowed events in every single type of date!
SeeMovieEvent
– name : String = "SeeMovie"
+ getName() :String
GoToRestaurantEvent
– name : String = "SeeMovie"
+ getName() :String
OrderFlowersEvent
– name : String = "SeeMovie"
+ getName() :String

How many classes did you have to touch to make Bob's changes? Five classes were changed or added to add just this one new type of event. First the "OrderFlowersEvent" class needed to be added, and then the method to order flowers on a date needed to be added to the Date class. Finally I had to update each of the different types of date to allow, or reject, the new type of event depending on whether it's allowed on that date or not

Are you happy with this design? Why or why not?

Five classes being changed seems like a LOT when all I'm adding is ONE new event. What happens when I have to add, say, a dozen new types of event; is it always going to involve this much work?

All done. It took a bit of work but we now have a Send Flowers event that you can add to a date.

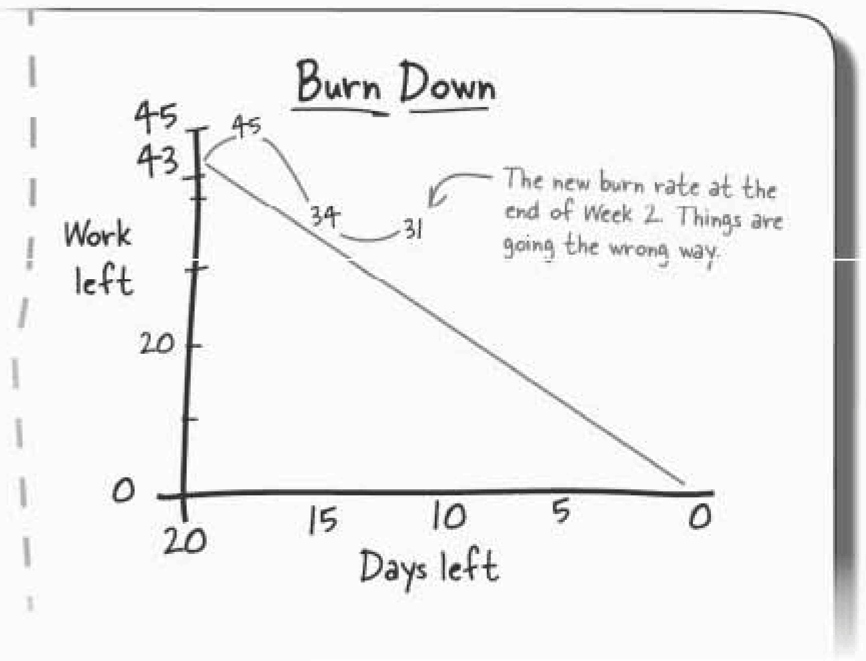
Laura: Hey, isn't "Buy jewelry" coming down the line? That works as just another event, too, right?

Bob: Yeah, but we'll need to add some time to make those changes to all the classes again.

Mark: Can't we come up with a more flexible design, so we can avoid this pain and effort each time we add a new event?

Bob: That's exactly what I was thinking.

Laura: But that will take even more time, right? I guess we're invested, though, huh? This will save us time later, I hope...



We interrupt this chapter...

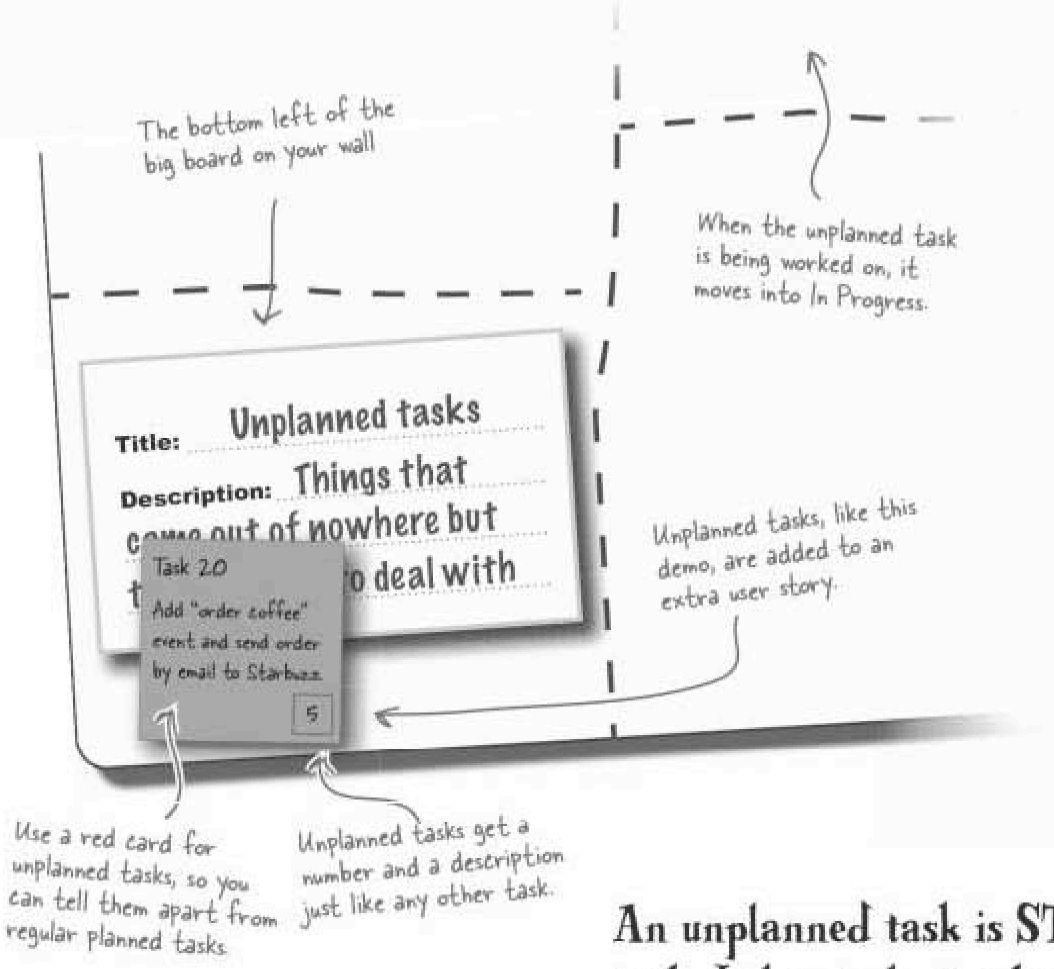
You're already getting behind on your burn-down rate and then the inevitable happens: the customer calls with a last-minute request...

Hey! The CEO of Starbuzz just called, and he wants to see a demo of ordering coffee as part of a date. Can you show me that tomorrow?
Your customer, iSwoon's CEO

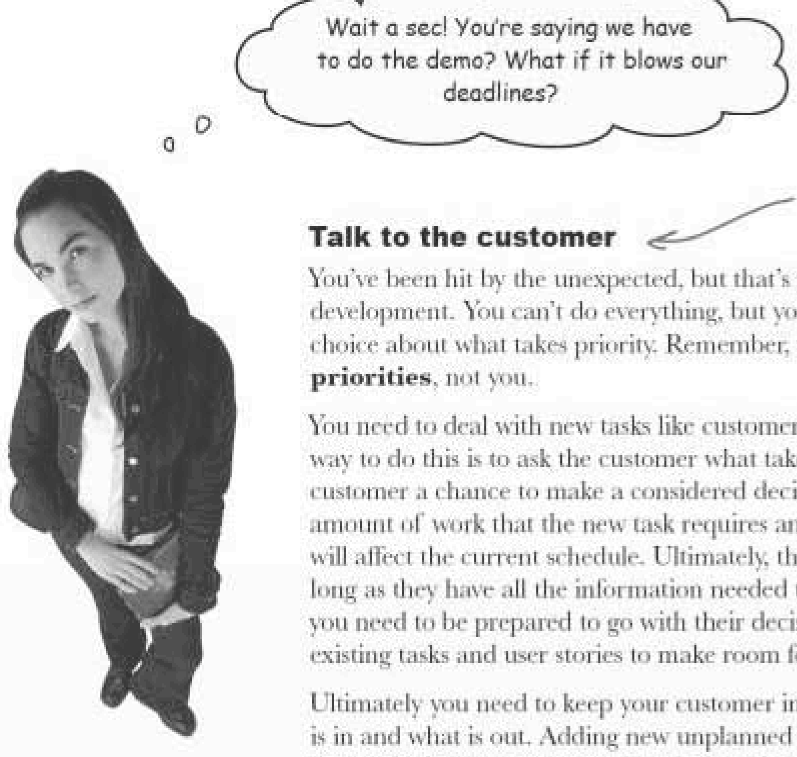
You have to track unplanned tasks

So far, your board has kept track of everything going on in your project. But what happens when somthing unplanned comes up? You have to track it, just like anything else. It affects your burn-down rate, the work you're doing on user stories, and more...

Let's take a look at a part of the board we haven't used yet:


The bottom left of the big board on your wall
When the unplanned task is being worked on, it moves into In Progress.
Title: Unplanned tasks
Description: Things that come out of nowhere but ... to deal with
Task 20
Add "order coffee" event and send order by email to Starbuzz.
5
Unplanned tasks, like this demo, are added to an extra user story.
Use a red card for unplanned tasks, so you can tell them apart from regular planned tasks.
Unplanned tasks get a number and a description just like any other task.

An unplanned task is STILL a task. It has to be tracked, put in progress, completed, and included in the burn-down rate just like <u>EVERY</u> <u>OTHER</u> <u>TASK</u> you have.


Wait a sec! You're saying we have to do the demo? What if it blows our deadlines?

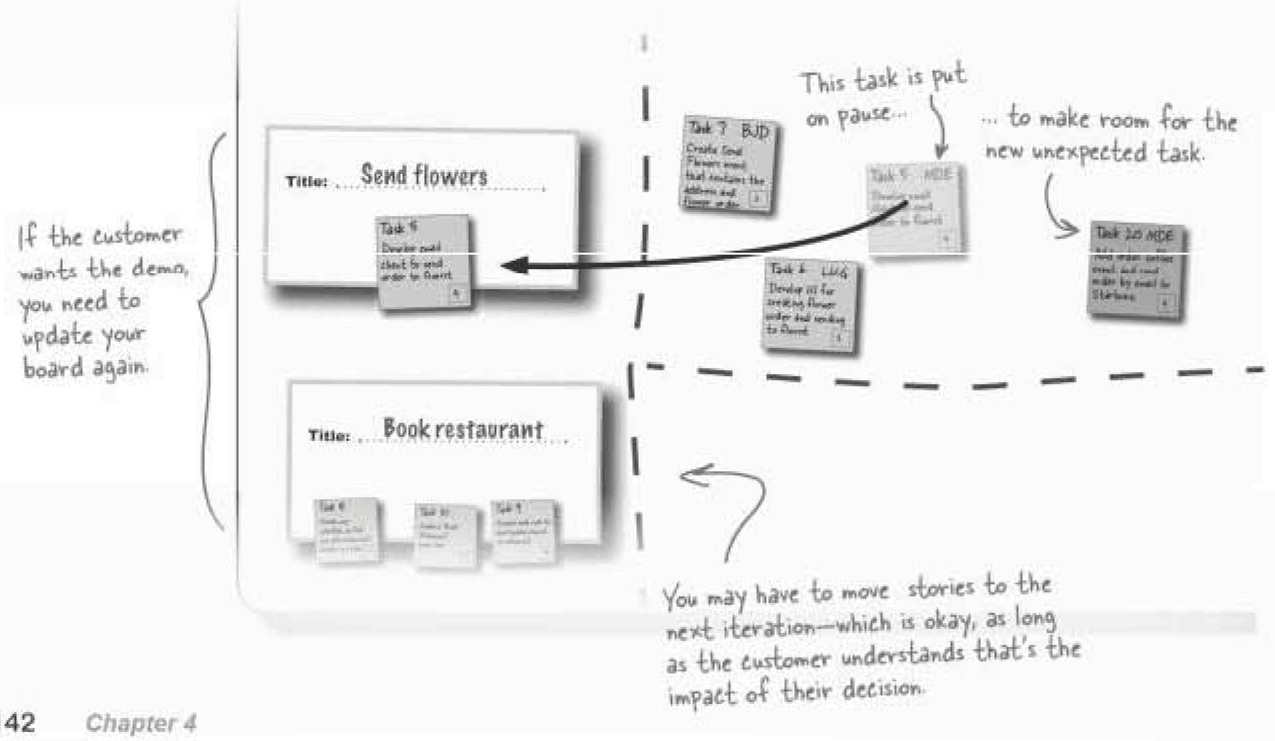
Yes, you've heard it before, but talking to the customer is the answer to most scheduling- and deadline-related problems.

Talk to the customer

You've been hit by the unexpected, but that's part of software development. You can't do everything, but you also can't make the choice about what takes priority. Remember, **the customer sets priorities**, not you.

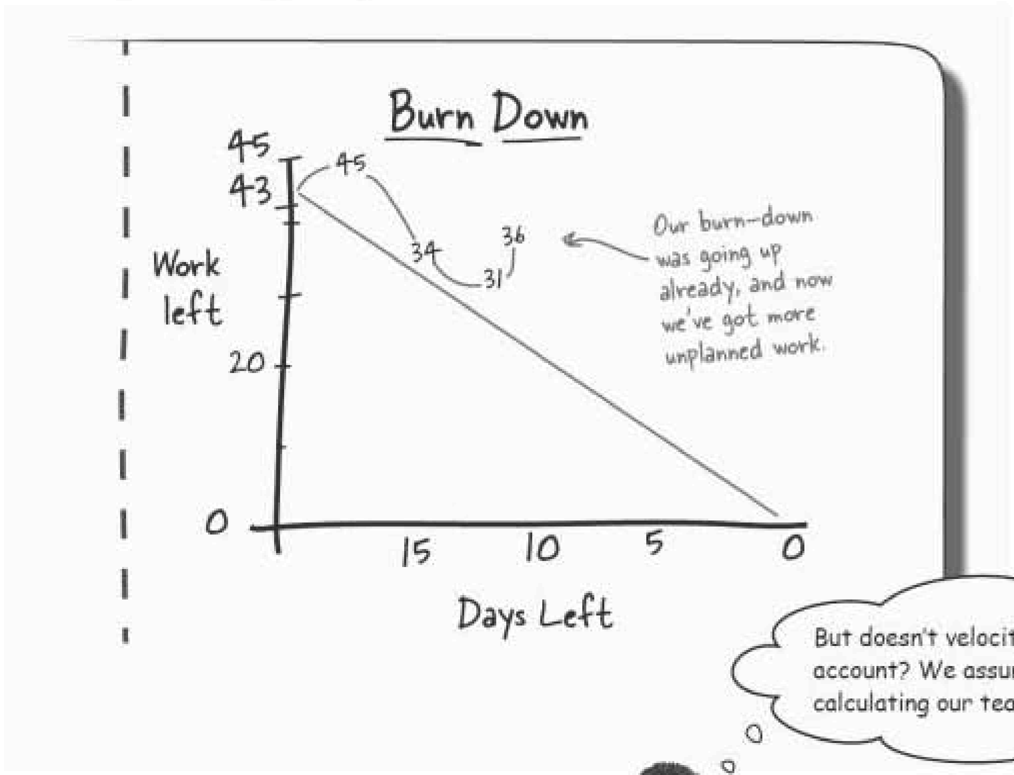
You need to deal with new tasks like customer demos, and the best way to do this is to ask the customer what takes priority. Give the customer a chance to make a considered decision by estimating the amount of work that the new task requires and explaining how that will affect the current schedule. Ultimately, the customer rules, so as long as they have all the information needed to make a choice, then you need to be prepared to go with their decision by reshuffling your existing tasks and user stories to make room for the surprise work.

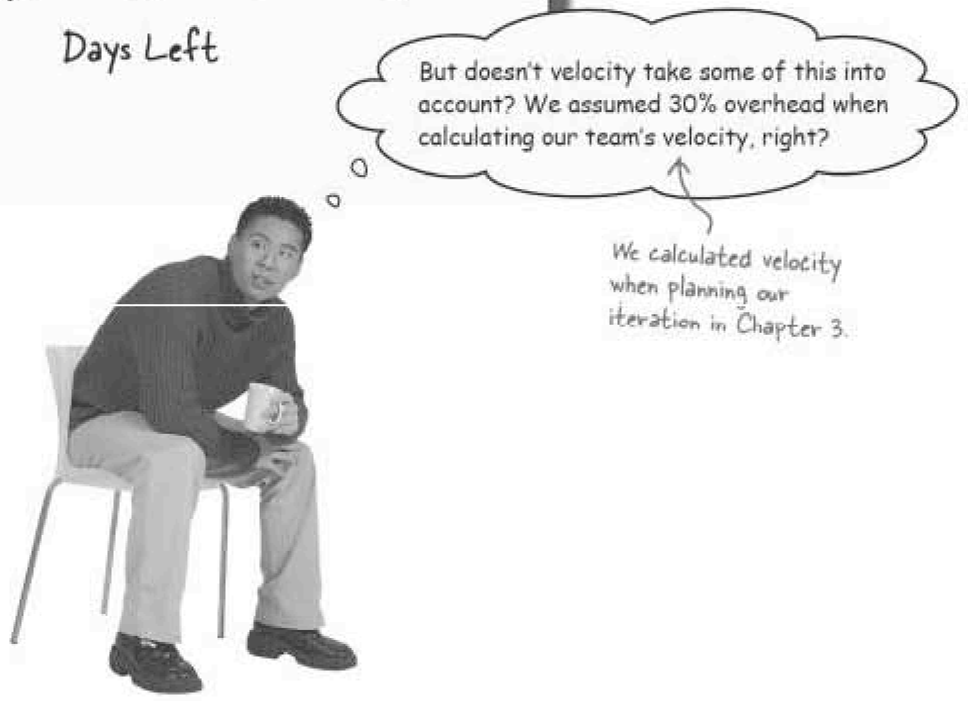
Ultimately you need to keep your customer in the picture as to what is in and what is out. Adding new unplanned work is not the end of the world, but your customer needs to understand that the work has an impact, and then they can choose what that impact is.


If the customer wants the demo, you need to update your board again.
Title: Send flowers
Title: Book restaurant
This task is put on pause...
... to make room for the new unexpected task.
Task 7 BJD
Task 6 LUG
Task 10 MDE
You may have to move stories to the next iteration—which is okay, as long as the customer understands that's the impact of their decision.

Unexpected tasks raise your burn-down rate

Unexpected task mean extra work. If the unexpected tasks can't be pushed into another iteration, then they need to be factored into your board. All of this means that your burn-down rate is affected, and not in a good way...




But doesn't velocity take some of this into account? We assumed 30% overhead when calculating our team's velocity, right?
We calculated velocity when planning our iteration in Chapter 3.

Velocity helps, but...

You've got more work thanks to some unexpected requirements from your customer, but didn't you factor this in when you calculated your team's velocity? Unfortunately, velocity is there to help you gauge how fast your team performs, but it's not there to handle unplanned tasks.

Remember this equation from Chapter 3?

We originally calculated velocity as...

$$3 \times 20 \times \underbrace{0.7}_{\text{Your team's first pass velocity, which is actually a guess at this point}} = 42$$

The number of people in your team

The amount of work in days that your team can handle in one iteration

So we have this much "float"...

$$3 \times 20 - 42 = 18$$

These are the possible days we could have, if everyone worked at 100% velocity...

... but it may not be enough!

~~Float—the "extra" days in your~~ schedule—disappear quickly.

An employee's car breaks down, someone has to go to the dentist, your daily standup meetings...those "extra" days disappear quickly. And remember, ***float is in work time, not actual time***. So if your company gives an extra Friday off for great work, that's *three* days of float lost because you are losing *three* developers for the whole day.

So when unplanned tasks come up, you may be able to absorb some of the extra time, but velocity won't take care of all of it.

So what do we do? This is major panic time, right? We're going to miss our deadlines...

there are no Dumb Questions

Q: You said to add unplanned tasks as red sticky notes. Do I have to use colored sticky notes? And why red?

A: We picked red because regular tasks are usually on regular yellow sticky notes, and because red stands out as a warning color. The idea is to quickly see what's part of your planned stories (the normal stickies), and what's unplanned (red). And red is a good "alert" color, since most unplanned tasks are high-priority (like that customer demo that came out of nowhere).

It's also important to know at the end of an iteration what you worked on.The red tasks make it easy to see what you dealt with that wasn't planned, so when you're recalculating velocity and seeing how good your estimates were, you know what was planned and what wasn't.

Q: So later on we're going to recalculate velocity?

A: Absolutely. Your team's velocity will be recalculated at the beginning of every single iteration. That way, you can get a realistic estimate of *your* team's productivity. 0.7 is just a good conservative place to start when you don't have any previous iterations to work from.

Q: So velocity is all about how me and my team performed in the last iteration?

A: Bingo. Velocity is a measure of how fast ***you*** and ***your team*** are working. The only way you can *reliably* come up with a figure for that is by looking at how well you performed in previous iterations.

Q: I really don't think 0.7 captures my team's velocity. Would it be OK to pick a faster or slower figure to start out with? Say 0.65, or 0.8?

A: You can pick a different starting velocity, but you have to stand by what you pick. If you know your team already at the beginning of a project, then it's perfectly alright to pick a velocity that matches your team's performance on other projects, although you should still factor in a slightly slower velocity at the beginning of any project. It always takes a little extra time to get your heads around what needs to be developed on a new project.

Remember, velocity is about how fast you and your team can comfortably work, for real. So you're aiming for a velocity that you believe in, and it's better to be slightly on the conservative side at the beginning of a new project, and then to refine that figure with hard data before each subsequent iteration.

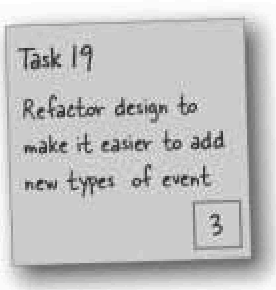
Velocity is NOT a substitute for good estimation; it's a way of factoring in the real-world performance of you and your team.

We have a lot to do...

You're in a tough spot. Doing some refactoring work is going to cost you time now, but the hope is that it will save you time in the long run. In addition you have the new demo that you need to prepare for the iSwoon CEO....

The "surprise" work that's needed for the demo that the CEO of iSwoon is giving to the CEO of Starbuzz. Talk about pressure!

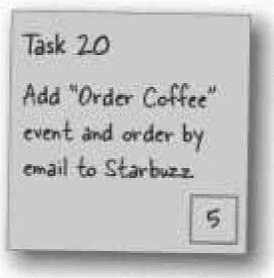
You've got more work to do...



Task 19

Refactor design to make it easier to add new types of event

3



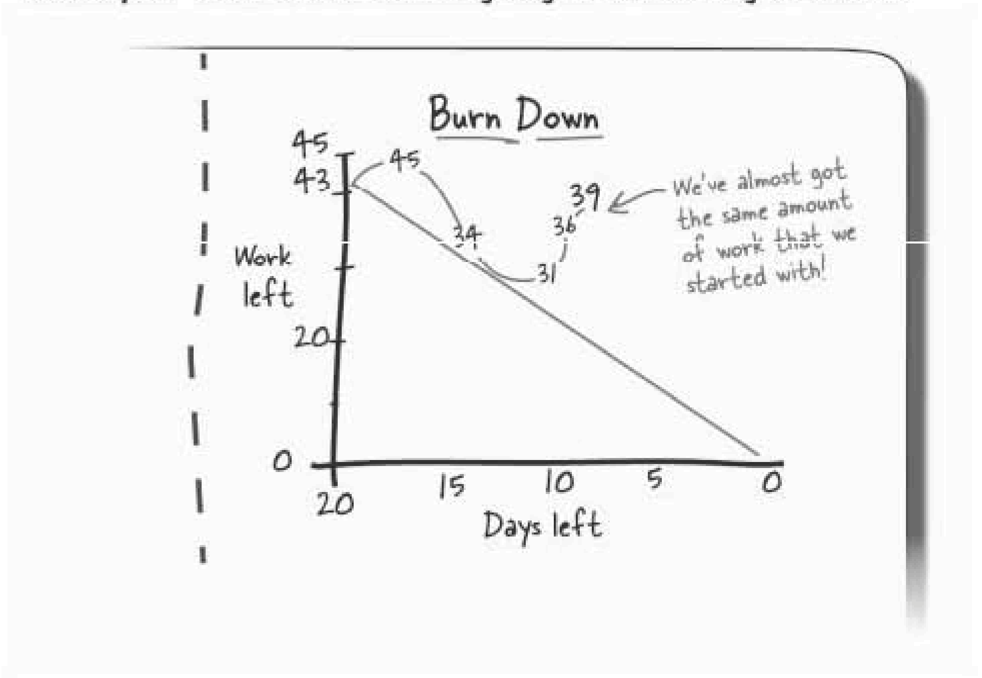
Task 20

Add "Order Coffee" event and order by email to Starbuzz

5

The task that came out of Bob's refactoring suggestion

...and your burn-down rate is going in the wrong direction.



...but we know EXACTLY where we stand

The customer knows where you are

At every step you've kept the customer involved so they know exactly what work they've added, and you can show them exactly what the changes will impact.

YOU know where you are

You and your development team are also on exactly the same page thanks to your board and the burn-down rate. This means that although things look a bit bleak, at least no one is burying their heads in the sand. The challenges are right there on your wall.

You know there are challenges, NOW.

Because you're monitoring your project using your board you know right now that there are challenges ahead if you're going to keep things on track. Compare this with the Big Bang "See you later, I'll deliver something in 3 months" approach from Chapter 1.

With the Big Bang approach, you didn't know you were in ~~trouble until day 30, or even day 90! With your board and~~ your burn-down rate you know immediately what you're facing, and that gives you the edge to make the calls to keep your development heading towards success.

Sometimes you'll hear this referred to as the waterfall approach

Successful software development is about knowing where you are.

With an understanding of your progress and challenges, you can keep your customer in the loop, and deliver software when it's needed.

All is far from lost! We'll tackle all these problems in Chapter 5, when we dig deeper into good class and application design, and handle the customer demo.

Velocity Exposed

This week's interview:
Keeping pace with Velocity

Head First: Welcome, Velocity, glad you could make time in your busy day to come talk with us.

Velocity: My pleasure, it's nice to be here.

Head First: So some would say that you have the potential to save a project that's in crisis, due perhaps to surprise changes or any of the other pieces of extra work that can hit a plan. What would you say to those people?

Velocity: Well, I'm really no superhero to be honest. I'm more of a safety net and confidence kinda guy.

Head First: What do you mean by "confidence"?

Velocity: I'm most useful when you're trying to come up with realistic plans, but not for dealing with the unexpected.

Head First: So you're really only useful at the beginning of a project?

Velocity: Well, I'm useful then, but at that point I'm usually just set to my default value of 0.7. My role gets much more interesting as you move from Iteration 1 to Iteration 2 and onwards.

Head First: And what do you offer for each iteration, confidence?

Velocity: Absolutely. As you move from one iteration to the next you can recalculate me to make sure that you can successfully complete the work you need to.

Head First: So you're more like a retrospective player?

Velocity: Exactly! I tell you how fast you were performing in the last iteration. You can then take that value and come up with a chunk of work in the next iteration that you can be much more confident that you can accomplish.

Head First: But when the unexpected comes along...

Velocity: Well, I can't really help too much with that, except that if you can increase your team's velocity, you might be able to fit in some more work. But that's a risky approach...

Head First: Risky because you really represent how fast your team works?

Velocity: That's exactly my point! I represent how fast your team works. If I say that you and your team, that's 3 developers total, can get 40 days of work done in an iteration, that's 20 work days long, that doesn't mean that there's 20 days there that you could possibly use if you just worked harder. Your team is always working as hard as they can, and I'm a measure of that. The danger is when people start using me as a pool of possible extra days of work...

Head First: So, if you could sum yourself up in one sentence, what would it be?

Velocity: I'm the guy that tells you how fast your team worked in the last iteration. I'm a measure of how you perform in reality, based on how you performed in the past, and I'm here to help you plan your iterations realistically.

Head First: Well, that's actually two sentences, but we'll let you get away with that. Thanks for making the time to come here today, Velocity.

Velocity: It's been a pleasure, nice to get some of these things off of my chest.